20070125 PACER 66 1st Amd Complaint-Ex F Pt 1.pdf

US006725356B2

(12) **United States Patent**
Hansen et al.

(10) **Patent No.: US 6,725,356 B2**
(45) **Date of Patent: *Apr. 20, 2004**

(54) **SYSTEM WITH WIDE OPERAND ARCHITECTURE, AND METHOD**

(75) Inventors: **Craig Hansen**, Los Altos, CA (US); **John Moussouris**, Palo Alto, CA (US)

(73) Assignee: **MicroUnity Systems Engineering, Inc.**, Sunnyvale, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **09/922,319**

(22) Filed: **Aug. 2, 2001**

(65) **Prior Publication Data**

US 2002/0133682 A1 Sep. 19, 2002

Related U.S. Application Data

(60) Continuation of application No. 09/382,402, filed on Aug. 24, 1999, now Pat. No. 6,295,599, which is a continuation-in-part of application No. 09/169,963, filed on Oct. 13, 1998, now Pat. No. 6,006,318, which is a continuation of application No. 08/754,827, filed on Nov. 22, 1996, now Pat. No. 5,822,603, which is a division of application No. 08/516,036, filed on Aug. 16, 1995, now Pat. No. 5,742,840.

(60) Provisional application No. 60/097,635, filed on Aug. 24, 1998.

(51) **Int. Cl.**[7] **G06F 15/00**

(52) **U.S. Cl.** **712/210**; 712/28; 712/24; 712/32; 712/208

(58) **Field of Search** 712/32, 28, 34, 712/208, 210, 20, 24

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,025,772 A 5/1977 Constant
4,489,393 A 12/1984 Kawahara et al.
4,701,875 A 10/1987 Konishi et al.

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

DE 0 654 733 A1 5/1994
JP 0 474 246 A2 6/1991

OTHER PUBLICATIONS

Parallel Computers for Graphics Applications, Adam Levinthal, Pat Hanrahan, Mike Paquette, Jim Lawson, Pixar San Rafael, California, 1987.
Organization of the Motorola 88110 Superscalar RISC Microprocessor, Keith Diefendorff and Michael Allen.
Microprocessor Report, vol. 7 No. 13, Oct. 4, 1993, IBM Regains Performance Lead with Power2, Six Way Superscalar CPU in MCM Achieves 126 SPECint92.

(List continued on next page.)

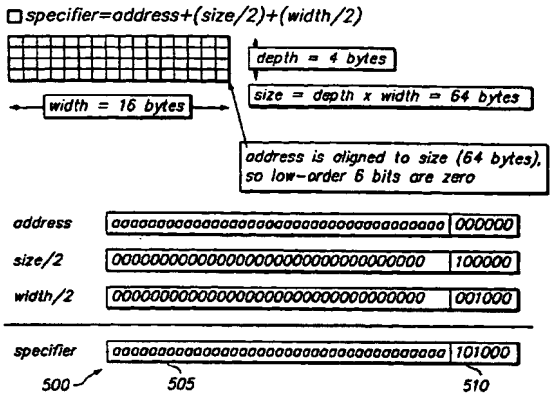
Primary Examiner—Matthew C. Bella
Assistant Examiner—Mackly Monestime
(74) *Attorney, Agent, or Firm*—McDermott, Will & Emery

(57) **ABSTRACT**

The present invention provides a system and method for improving the performance of general purpose processors by expanding at least one source operand to a width greater than the width of either the general purpose register or the data path width. In addition, the present invention provides several classes of instructions which cannot be performed efficiently if the operands are limited to the width and accessible number of general purpose registers. The present invention provides operands which are substantially larger than the data path width of the processor by using a general purpose register to specify a memory address from which at least more than one, but typically several data path widths of data can be read. The present invention also provides for the efficient usage of a multiplier array that is fully used for high precision arithmetic, but is only partly used for other, lower precision operations.

48 Claims, 148 Drawing Sheets

Microfiche Appendix Included
(5 Microfiche, 63 Pages)



U.S. PATENT DOCUMENTS

4,727,505 A		2/1988	Konishi et al.	
4,876,660 A		10/1989	Owens et al.	
4,893,267 A		1/1990	Alsup et al.	
4,956,801 A		9/1990	Priem et al.	
4,969,118 A		11/1990	Montoye et al.	
4,975,868 A		12/1990	Freerksen	
5,032,865 A		7/1991	Schlunt	
5,157,388 A		10/1992	Kohn	
5,201,056 A		4/1993	Daniel et al.	
5,268,855 A		12/1993	Mason et al.	
5,268,995 A		12/1993	Diefendorff et al.	
5,280,598 A	*	1/1994	Osaki et al.	710/127
5,408,581 A		4/1995	Suzuki et al.	
5,423,051 A		6/1995	Fuller et al.	
5,426,600 A		6/1995	Nakagawa et al.	
5,487,024 A	*	1/1996	Girardeau, Jr.	708/606
5,500,811 A		3/1996	Corry	
5,557,724 A		9/1996	Sampat et al.	
5,588,152 A		12/1996	Dapp et al.	
5,592,405 A		1/1997	Gove et al.	
5,600,814 A	*	2/1997	Gahan et al.	711/100
5,640,543 A		6/1997	Farrell et al.	
5,642,306 A		6/1997	Mennemeier et al.	
5,666,298 A		9/1997	Peleg et al.	
5,669,010 A		9/1997	Duluk, Jr.	
5,673,407 A		9/1997	Poland et al.	
5,675,526 A		10/1997	Peleg et al.	
5,680,338 A		10/1997	Agarwal et al.	
5,721,892 A		2/1998	Peleg et al.	
5,734,874 A		3/1998	Van Hook et al.	
5,757,432 A		5/1998	Dulong et al.	
5,758,176 A		5/1998	Agarwal et al.	
5,768,546 A	*	6/1998	Kwon	710/127
5,802,336 A		9/1998	Peleg et al.	
5,809,292 A		9/1998	Wilkinson et al.	
5,818,739 A		10/1998	Peleg et al.	
5,825,677 A		10/1998	Agarwal et al.	
5,835,782 A		11/1998	Lin et al.	
5,886,732 A		3/1999	Humpleman	
5,922,066 A		7/1999	Cho et al.	
5,983,257 A		11/1999	Dulong et al.	
6,016,538 A		1/2000	Guttag et al.	
6,092,094 A		7/2000	Ireton	
6,295,599 B1	*	9/2001	Hansen et al.	712/32
6,401,194 B1		6/2002	Nguyen et al.	

OTHER PUBLICATIONS

IBM Creates PowerPC Processors for AS/400, Two New CPU's Implement 64–Bit Power PC with Extensions by Linley Gwennap, Jul. 31, 1995.

The Visual Instruction Set (VSI) in UltraSPAR™, L. Kohn, G. Maturana, M. Tremblay, A. Prabhu, G. Zyner, May 3, 1995.

Osborne McGraw–Hill, i860™ Microprocessor Architecture, Neal Margulis, Foreword by Les Kohn.

A General–Purpose Array Processor for Seismic Processing, Nov.–Dec., 1984, vol. 1, No. 3) Revisiting past digital signal processor technology, Don Shaver– Jan.–Mar. 1998.

Accelerating Multimedia with Enhanced Microprocessors, Ruby B. Lee, 1995.

* cited by examiner

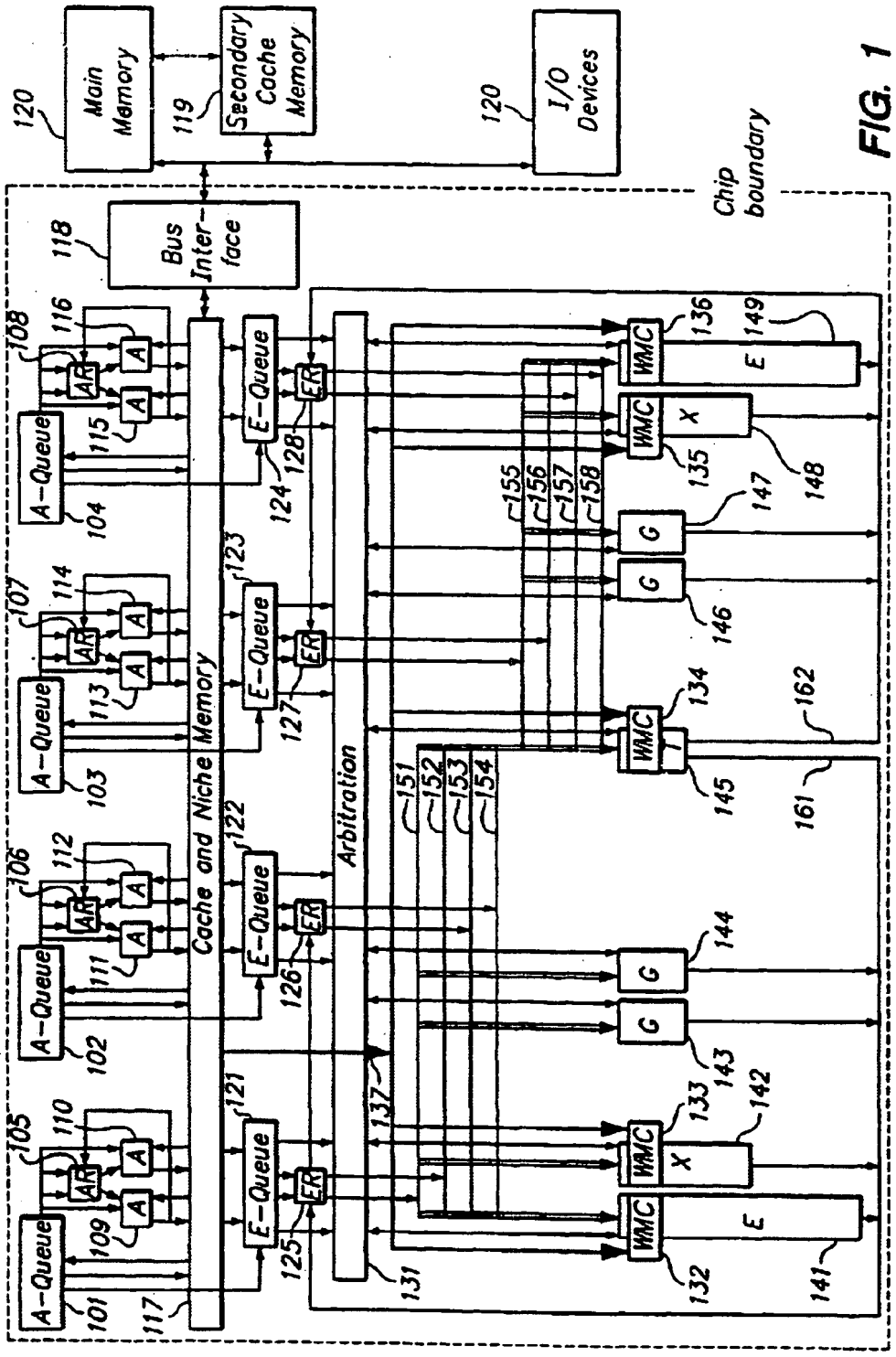

Main Memory
Secondary Cache Memory
I/O Devices
Chip boundary
Bus Interface
Cache and Niche Memory
Arbitration
A-Queue
E-Queue
AR
A
ER
WMC
G
X
E

FIG. 1

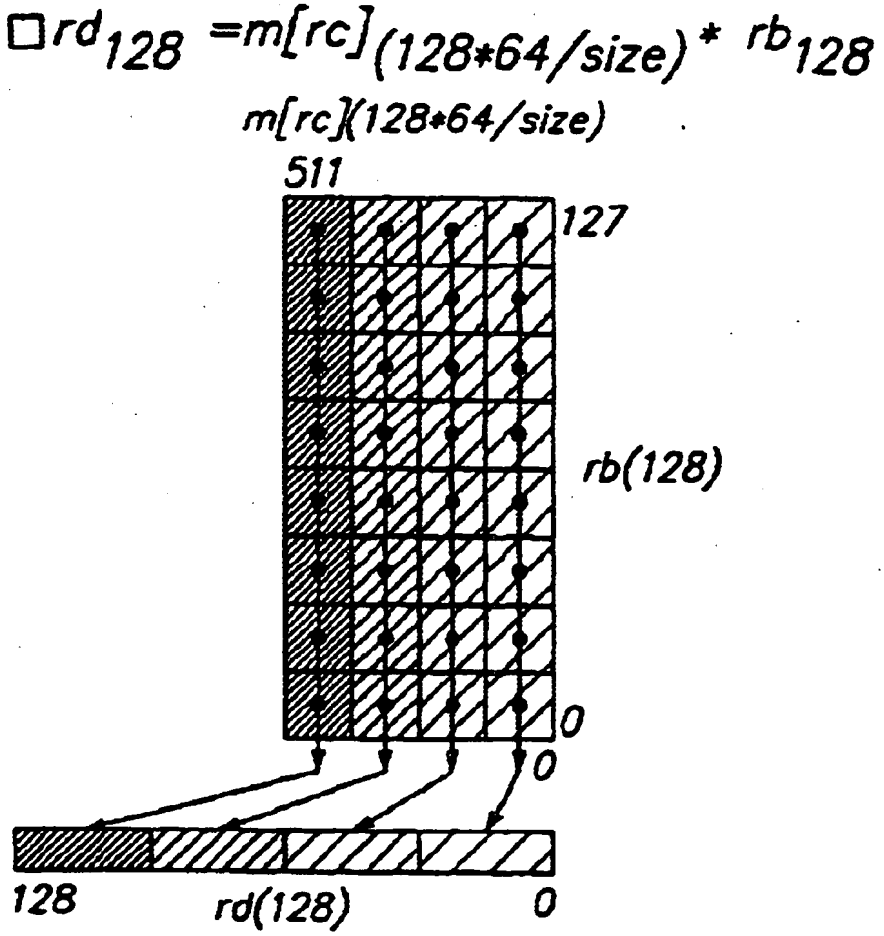

rd128 =m[rc](128*64/size) * rb128
m[rc](128*64/size)
511
127
rb(128)
0
0
128
rd(128)
0

FIG. 2

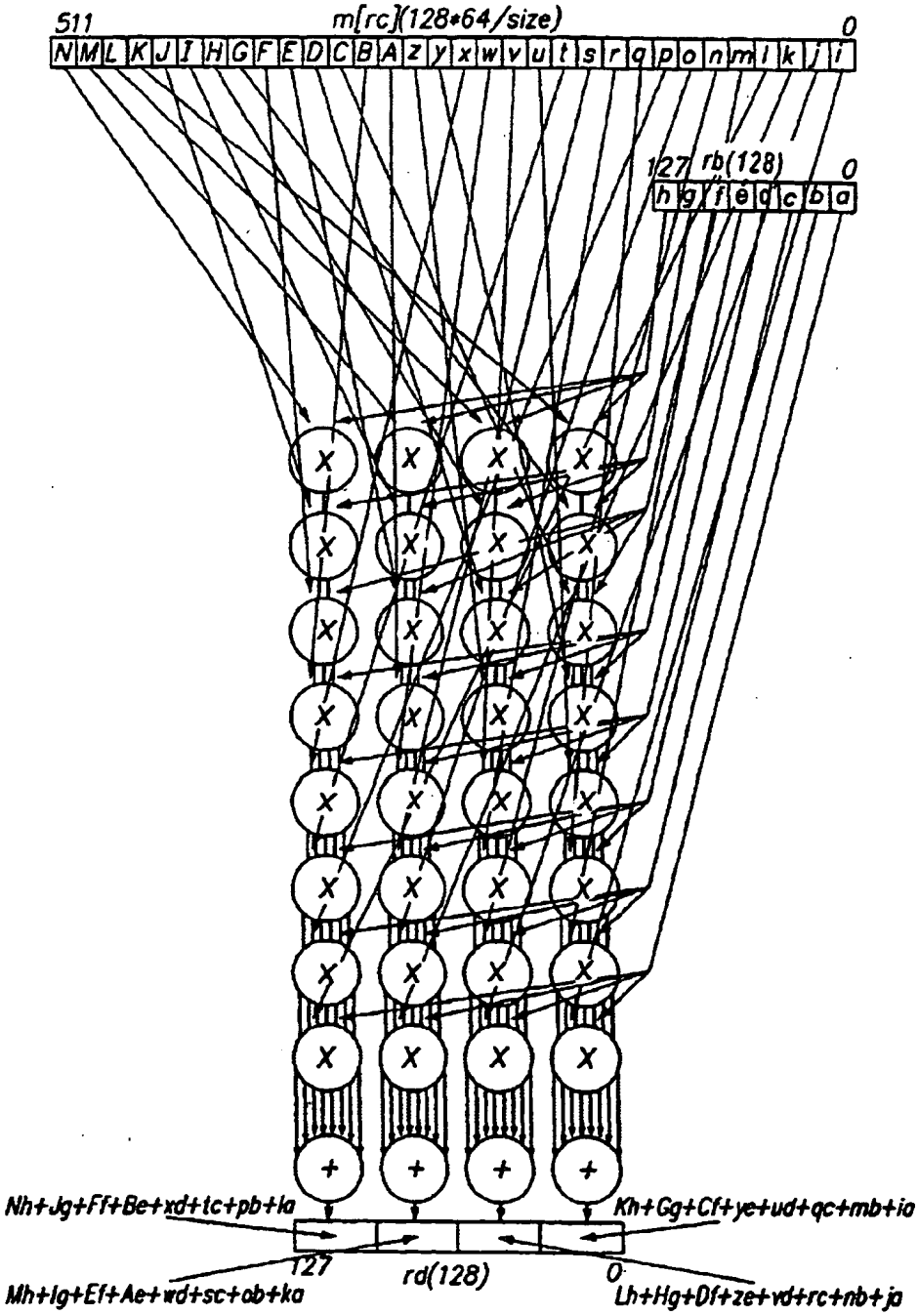

511
m[rc](128*64/size)
0
N M L K J I H G F E D C B A z y x w v u t s r q p o n m l k j i
127
rb(128)
0
h g f e d c b a
Nh+Jg+Ff+Be+xd+tc+pb+ia
Kh+Gg+Cf+ye+ud+qc+mb+io
127
rd(128)
0
Mh+Ig+Ef+Ae+wd+sc+ob+ka
Lh+Hg+Df+ze+vd+rc+nb+ja

FIG. 3

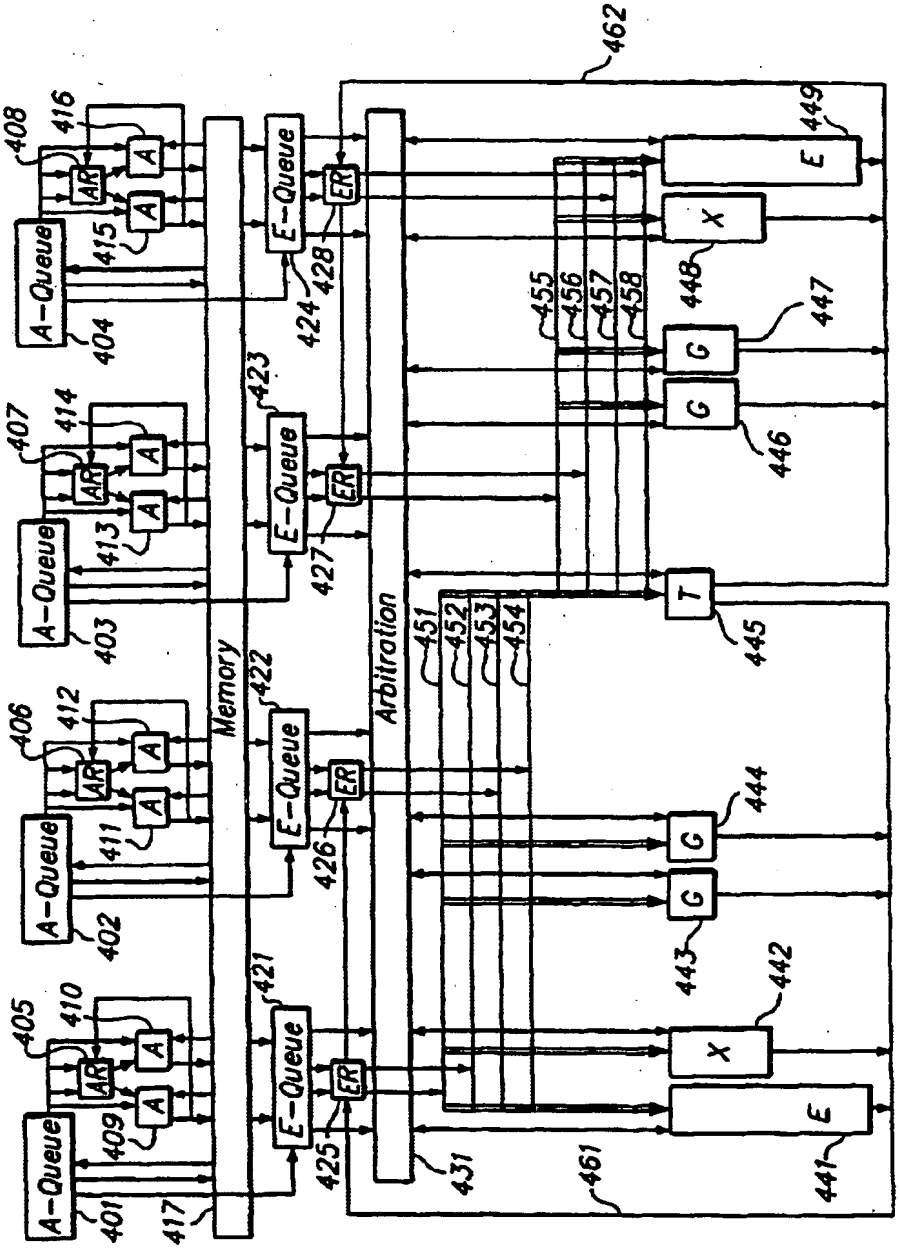

462
A-Queue
E-Queue
Memory
Arbitration
AR
A
ER
E
X
G
T
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
421
422
423
424
425
426
427
428
431
441
442
443
444
445
446
447
448
449
451
452
453
454
455
456
457
458
461

FIG. 4

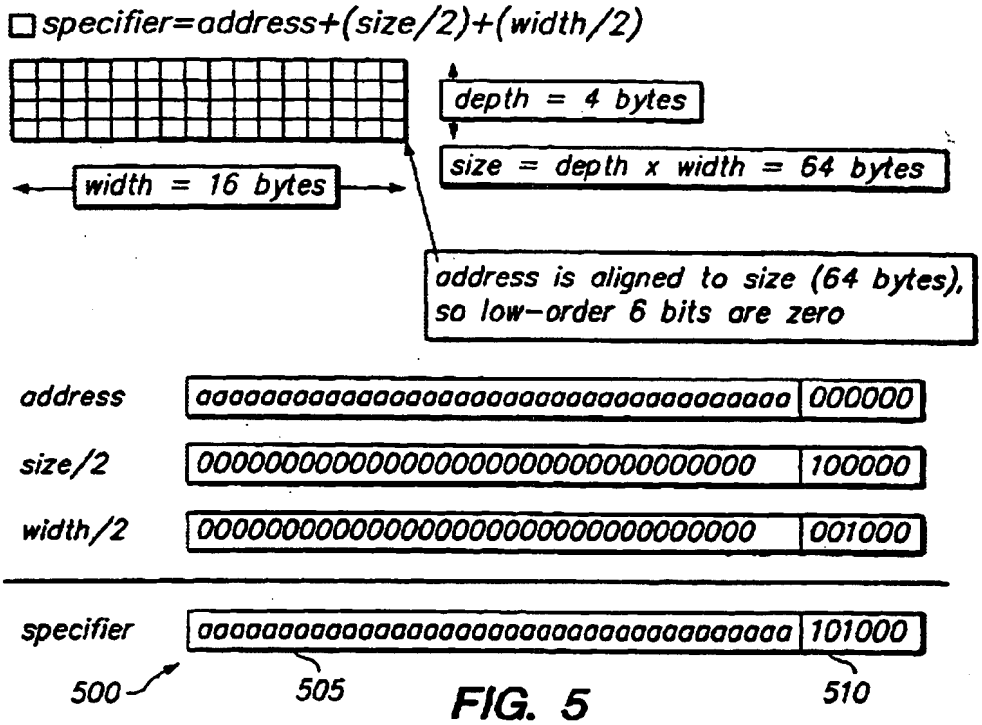

specifier=address+(size/2)+(width/2)
depth = 4 bytes
size = depth x width = 64 bytes
width = 16 bytes
address is aligned to size (64 bytes), so low-order 6 bits are zero
address
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
000000
size/2
00000000000000000000000000000000
100000
width/2
00000000000000000000000000000000
001000
specifier
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
101000
500
505
510

FIG. 5

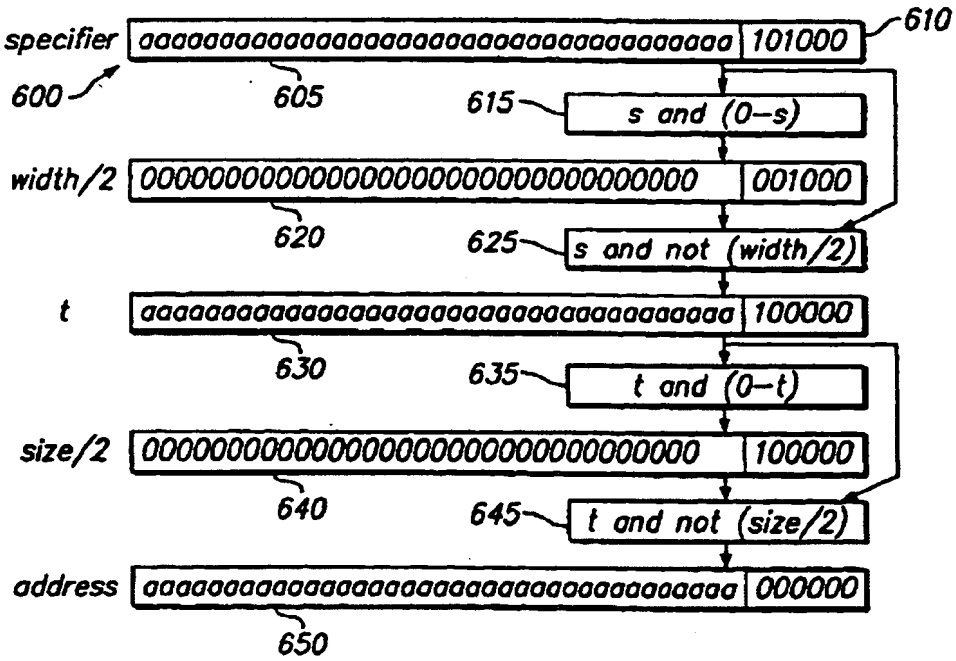

specifier
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
101000
610
600
605
615
s and (0-s)
width/2
00000000000000000000000000000000
001000
620
625
s and not (width/2)
t
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
100000
630
635
t and (0-t)
size/2
00000000000000000000000000000000
100000
640
645
t and not (size/2)
address
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
000000
650

FIG. 6

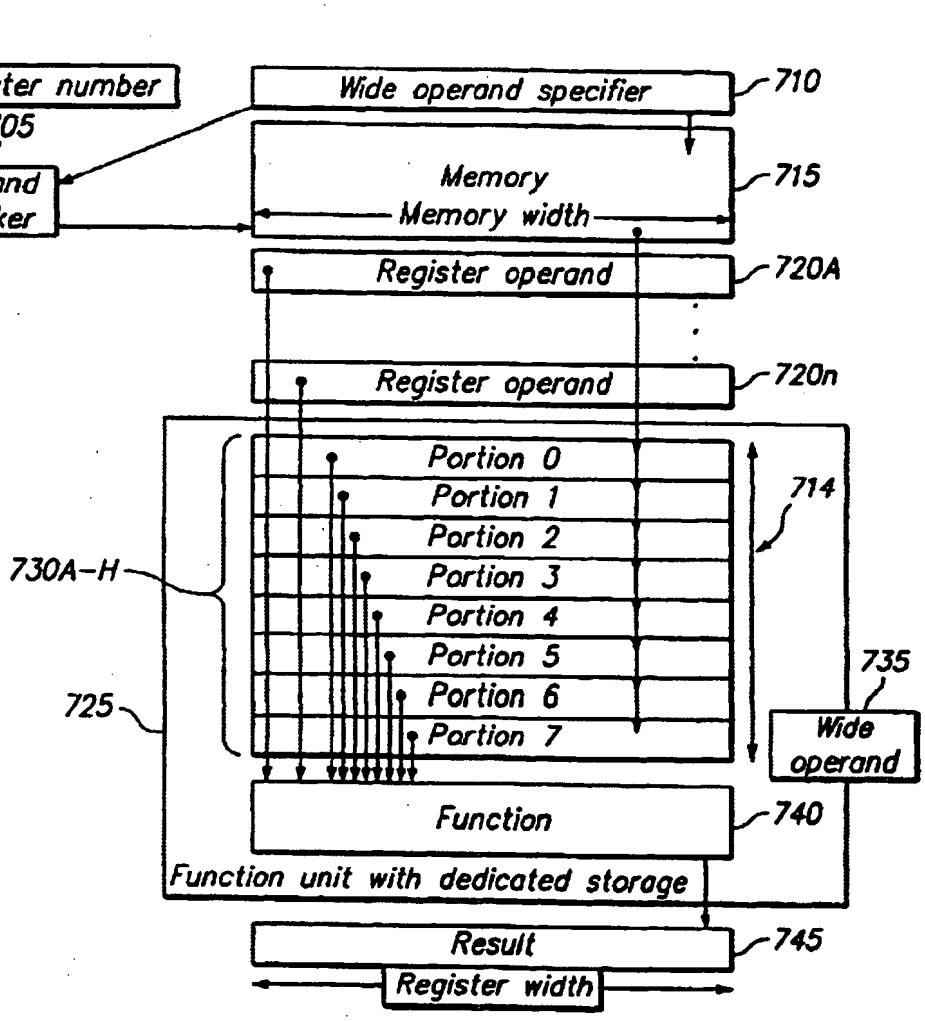

700
Register number
705
Operand checker
Wide operand specifier
710
Memory
Memory width
715
Register operand
720A
Register operand
720n
Portion 0
Portion 1
Portion 2
Portion 3
Portion 4
Portion 5
Portion 6
Portion 7
714
730A-H
725
735
Wide operand
Function
740
Function unit with dedicated storage
Result
745
Register width

FIG. 7

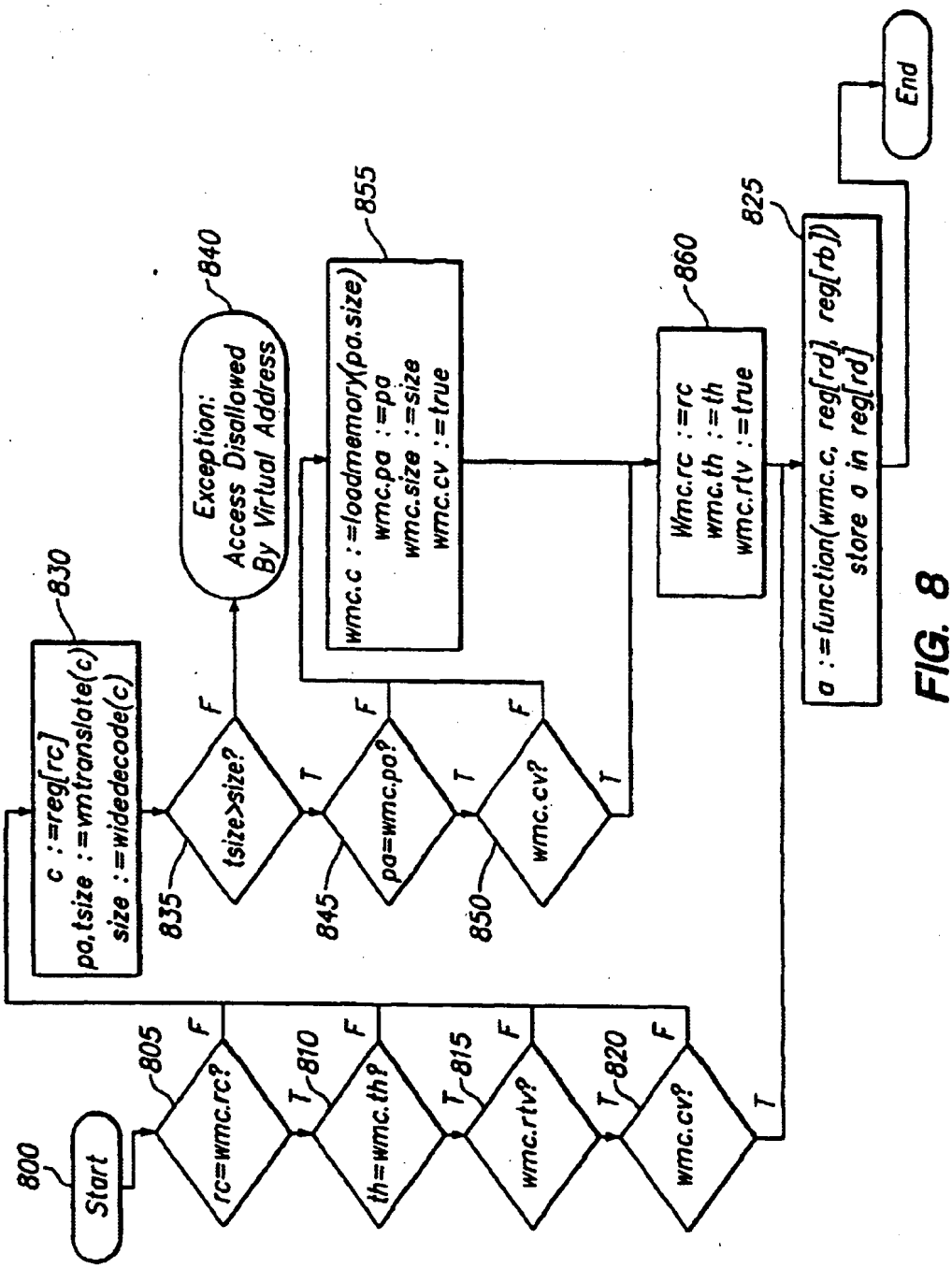

800
Start
805
rc=wmc.rc?
810
th=wmc.th?
815
wmc.rtv?
820
wmc.cv?
825
a :=function(wmc.c, reg[rd], reg[rb])
store a in reg[rd]
End
830
c :=reg[rc]
pa,tsize :=vmtranslate(c)
size :=widedecode(c)
835
tsize>size?
840
Exception:
Access Disallowed
By Virtual Address
845
pa=wmc.pa?
850
wmc.cv?
855
wmc.c :=loadmemory(pa,size)
wmc.pa :=pa
wmc.size :=size
wmc.cv :=true
860
Wmc.rc :=rc
wmc.th :=th
wmc.rtv :=true
T
F

FIG. 8

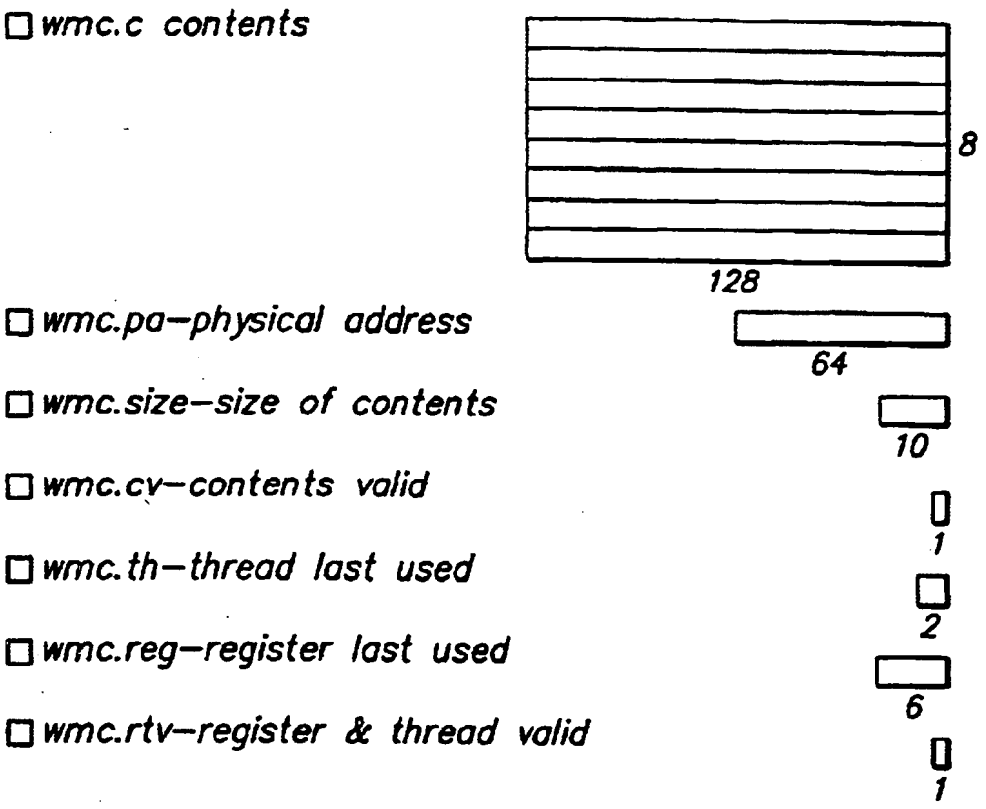

wmc.c contents
8
128
wmc.pa–physical address
64
wmc.size–size of contents
10
wmc.cv–contents valid
1
wmc.th–thread last used
2
wmc.reg–register last used
6
wmc.rtv–register & thread valid
1

FIG. 9

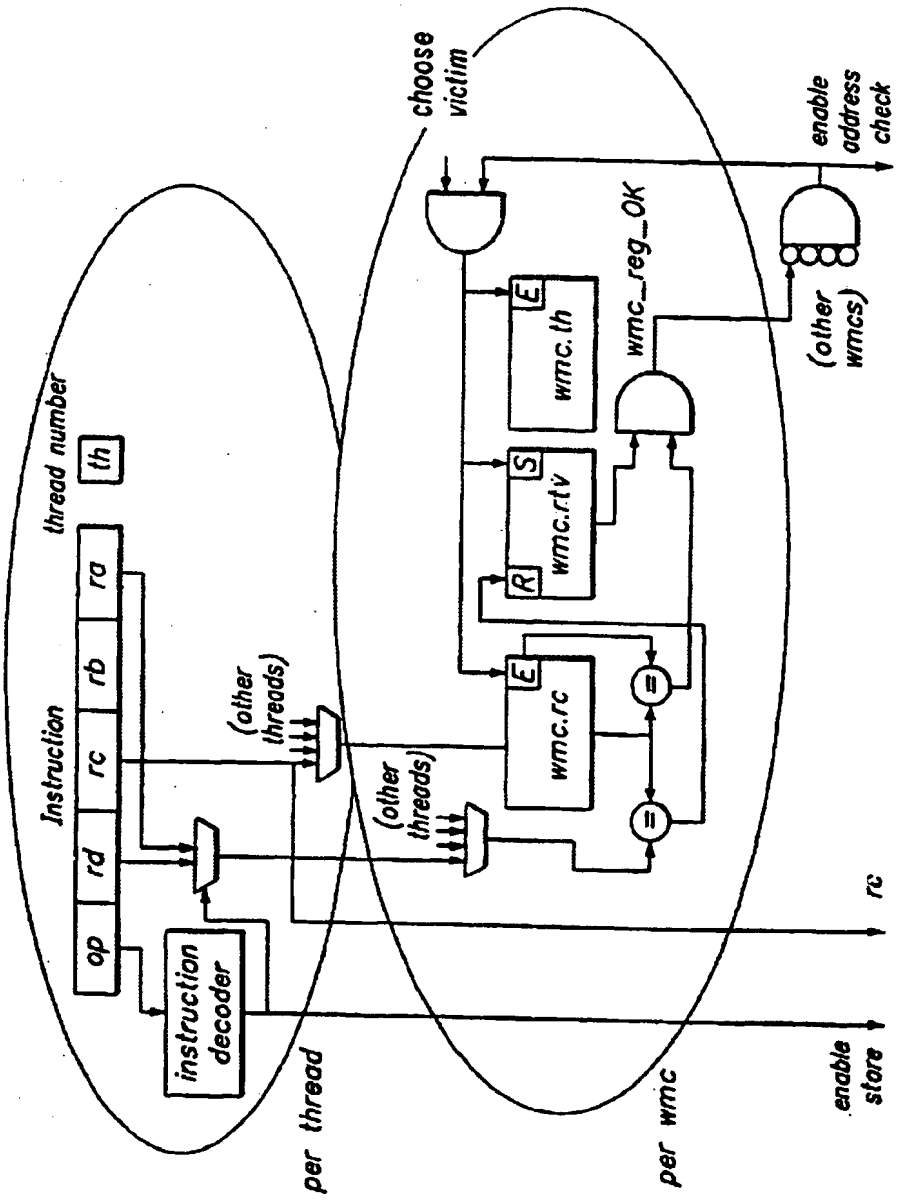

choose victim
enable address check
wmc_reg_OK
(other wmcs)
wmc.th
wmc.rtv
wmc.rc
thread number
th
Instruction
ra
rb
rc
rd
op
(other threads)
(other threads)
instruction decoder
rc
enable store
per thread
per wmc

FIG. 10

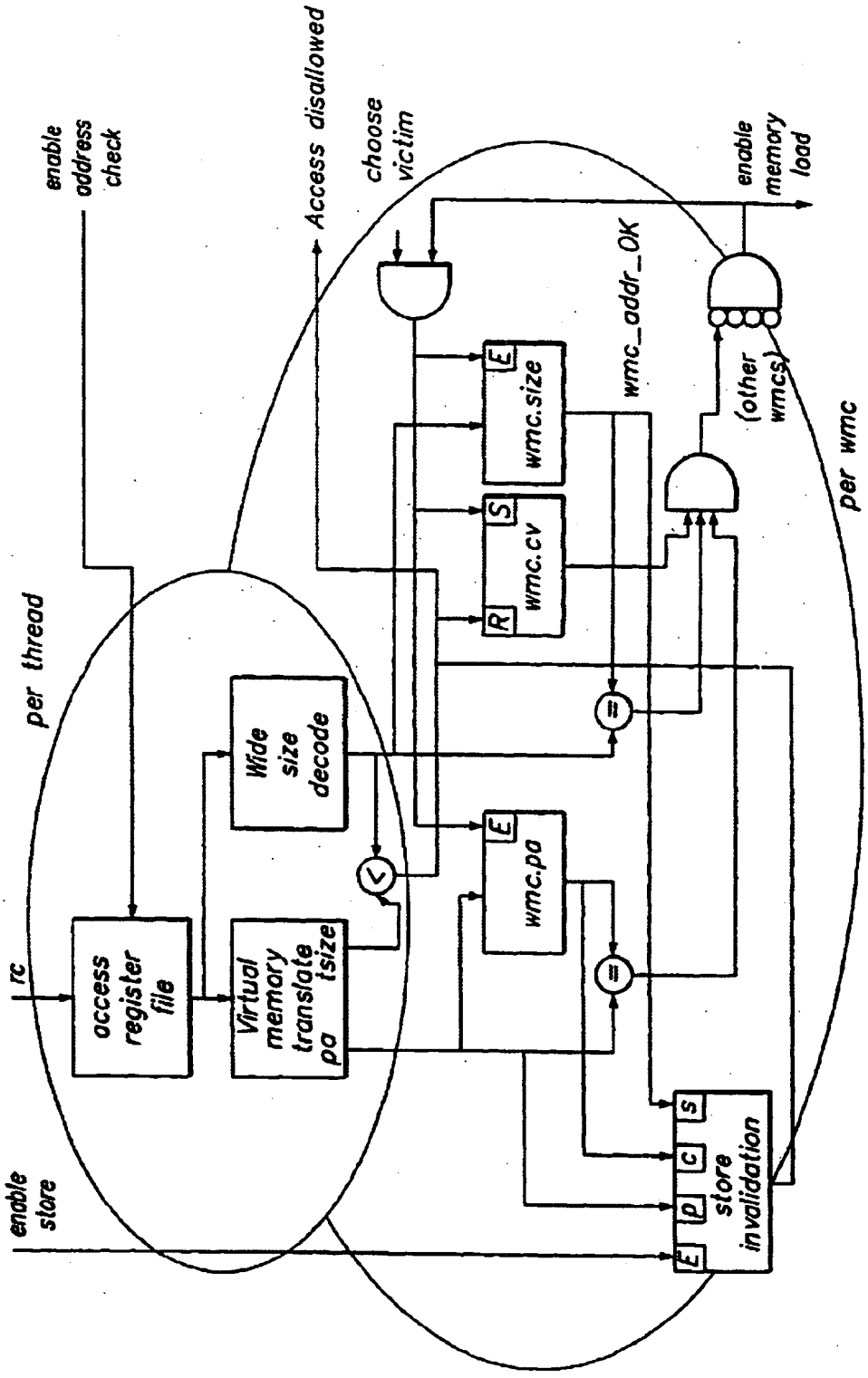

enable address check
Access disallowed
choose victim
enable memory load
wmc_addr_OK
(other wmcs)
per wmc
per thread
E wmc.size
S wmc.cv
R
E wmc.pa
Wide size decode
rc
access register file
Virtual memory translate tsize pa
enable store
E
D
C
S
store invalidation

FIG. 11

210

Operation codes

W.SWITCH.B	Wide switch big-endian
W.SWITCH.L	Wide switch little-endian

Selection

class	op	order
Wide switch	W.SWITCH	B L

Format

W.op.order ra=rc,rd,rb

ra=woporder(rc,rd,rb)

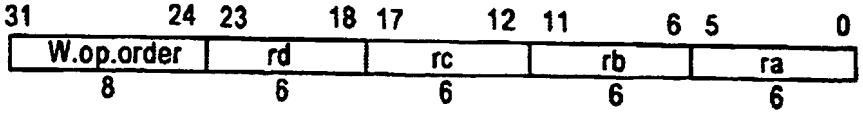


FIG. 12A

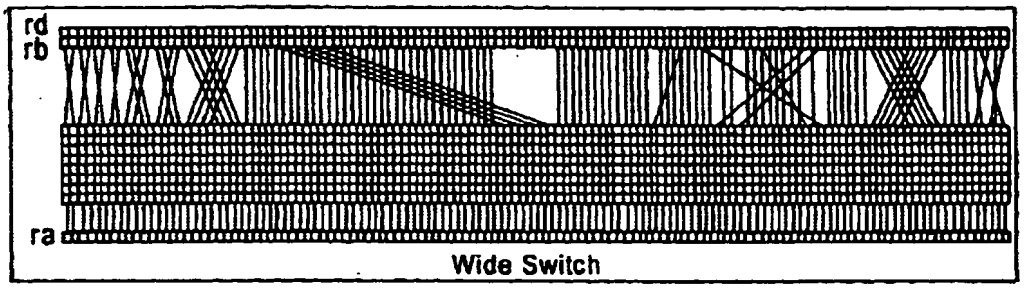
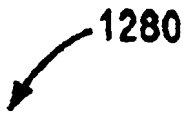


FIG. 12B

1250

```
Definition
defWideSwitch(op,rd,rc,rb,ra)
    d ← RegRead(rd, 128)
    c ← RegRead(rc, 64)
    b ← RegRead(rb, 128)
    if c1..0 ≠ 0 then
            raise AccessDisallowedByVirtual Address
    elseif c6..0 ≠0 then
            VirtAddr ← c and (c-1)
            W ← wsize ← (c and (0-c)) || 0^1
    else
            VirAddr ← c
            w ← wsize ← 128
    endif
    msize ← 8*wsize
    lwsize ← log(wsize)
    case op of
            W.SWITCH.B:
                order ← B
            W.SWITCH.L:
                order ← L
    endcase
    m ← LoadMemory(c, VirtAddr,msize,order)
    db ← d || b
    for i ← 0 to 127
            j ← 0|| i1wsize-1..0
            k ← m7*w+j||m6*w+j||m5*w+j||m4*w+j||m3*w+j||m2*w+j||mw+j||mj
            1 ← i7..1wsize||j1wsize-1..0
            ai ← db1
     endfor
     RegWrite(ra, 128, a)
enddef
```

FIG. 12C



Exceptions

Access disallowed by virtual address
Access disallowed by tag
Access disallowed by global TB
Access disallowed by local TB
Access detail required by tag
Access detail required by local TB
Access detail required by global TB
Local TB miss
Global TB miss

FIG. 12D

1210

Operation codes

W.TRANSLATE.8.B	Wide translate bytes big-endian
W.TRANSLATE.16.B	Wide translate doublets bit-endian
W.TRANSLATE.32.B	Wide translate quadlets bit-endian
W.TRANSLATE.64.B	Wide translate octlets big-endian
W.TRANSLATE.8.L	Wide translate bytes little-endian
W.TRANSLATE.16.L	Wide translate doublets little-endian
W.TRANSLATE.32.L	Wide translate quadlets little-endian
W.TRANSLATE.64.L	Wide translate octlets little-endian

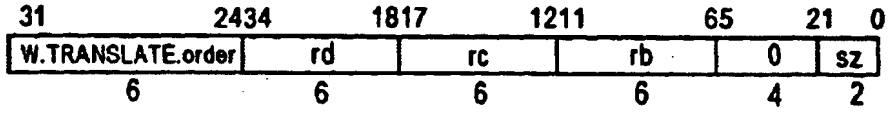
Selection

class	size	order
Wide translate	8 16 32 64	B L

Format

W.TRANSLATE.size.order rd=rc,rb

rd=wtranslatesizeorder(rc,rb)



sz ← log(size) = 3

FIG. 13A

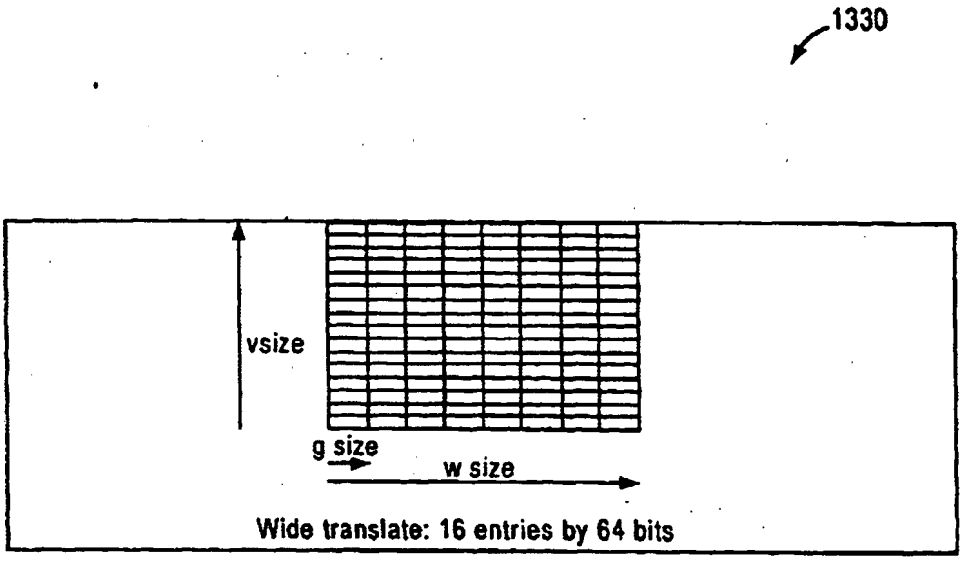


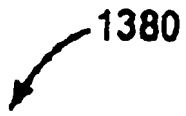
FIG. 13B

1350

Definition

```
def Wide Translate(op,gsize,rd,rc,rb)
    c←RegRead(rc, 64)
    b←RegRead(rb, 128)
    lgsize←log(gsize)
    if c_{lgsize-4..0} ≠ 0 then
        raise AccessDisallowedByVirtual Address
    endif
    if c_{4..lgsize-3} ≠ 0 then
        wsize←(c and (0-c)) || 0^3
        t←c and (c-1)
    else
        wsize←128
        t←c
    endif
    lwsize←log(wsize)
if t_{lwsize+4..lwsize-2} ≠ 0 then
        msize←(t and (0-t)) || 0^4
        VirtAddr←t and (t-1)
    else
        msize←256*wsize
        VirtAddr←t
    endif
    case op of
        W.TRANSLATE.B:
            order←B
        W.TRANSLATE.L:
            order←L
    endcase
    m←LoadMemory(c,VirtAddr,msize,order)
    vsize←msize/wsize
    lvsize←log(vsize)
    for i←0 to 128-gsize by gsize
        j←((order=B)^lvsize )^(b_{lvsize-1+i..i} ))*wsize+i_{lwsize-1..0}
        a_{gsize-1+i..i}←m_{j+gsize-1..j}
    endfor
    RegWrite(rd, 128, a)
enddef
```

FIG. 13C



Exceptions

Access disallowed by virtual address
Access disallowed by tag
Access disallowed by global TB
Access disallowed by local TB
Access detail required by tag
Access detail required by local TB
Access detail required by global TB
Local TB miss
Global TB miss

FIG. 13D

Operation codes ← 1410

W.MUL.MAT.8.B	Wide multiply matrix signed byte big-endian
W.MUL.MAT.8.L	Wide multiply matrix signed byte little-endian
W.MUL.MAT.16.B	Wide multiply matrix signed doublet big-endian
W.MUL.MAT.16.L	Wide multiply matrix signed doublet little-endian
W.MUL.MAT.32.B	Wide multiply matrix signed quadlet big-endian
W.MUL.MAT.32.L	Wide multiply matrix signed quadlet little-endian
W.MUL.MAT.C.8.B	Wide multiply matrix signed complex byte big-endian
W.MUL.MAT.C.8.L	Wide multiply matrix signed complex byte little-endian
W.MUL.MAT.C.16.B	Wide multiply matrix signed complex doublet big-endian
W.MUL.MAT.C.16.L	Wide multiply matrix signed complex doublet little-endian
W.MUL.MAT.M.8.B	Wide multiply matrix mixed-signed byte big-endian
W.MUL.MAT.M.8.L	Wide multiply matrix mixed-signed byte little-endian
W.MUL.MAT.M.16.B	Wide multiply matrix mixed-signed doublet big-endian
W.MUL.MAT.M.16.L	Wide multiply matrix mixed-signed doublet little-endian
W.MUL.MAT.M.32.B	Wide multiply matrix mixed-signed quadlet big-endian
W.MUL.MAT.M.32.L	Wide multiply matrix mixed-signed quadlet little-endian
W.MUL.MAT.P.8.B	Wide multiply matrix polynomial byte big-endian
W.MUL.MAT.P.8.L	Wide multiply matrix polynomial byte little-endian
W.MUL.MAT.P.16.B	Wide multtply matrix polynomial doublet big-endian
W.MUL.MAT.P.16.L	Wide multiply matrix polynomial doublet little-endian
W.MUL.MAT.P.32.B	Wide multiply matrix polynomial quadlet big-endian
W.MUL.MAT.P.32.L	Wide multiply matrix polynomial quadlet little-endian
W.MUL.MAT.U.8.B	Wide multiply matrix unsigned byte big-endian
W.MUL.MAT.U.8.L	Wide multiply matrix unsigned byte little-endian
W.MUL.MAT.U.16.B	Wide multiply matrix unsigned doublet big-endian
W.MUL.MAT.U.16.L	Wide multiply matrix unsigned doublet little-endian
W.MUL.MAT.U.32.B	Wide multiply matrix unsigned quadlet big-endian
W.MUL.MAT.U.32.L	Wide multiply matrix unsigned quadlet little-endian

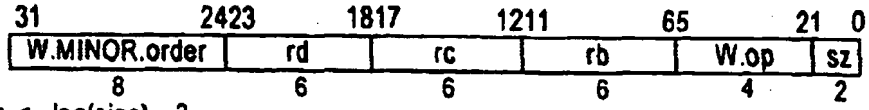
Selection

class	op	type	size	order
multiply	W.MUL.MAT	NONE MUP	8 16 32	B L
		C	8 16	B L

Format

W.op.size.order rd=rc,rb

rd=wopsizeorder(rc,rb)



sz ← log(size) - 3

FIG. 14A

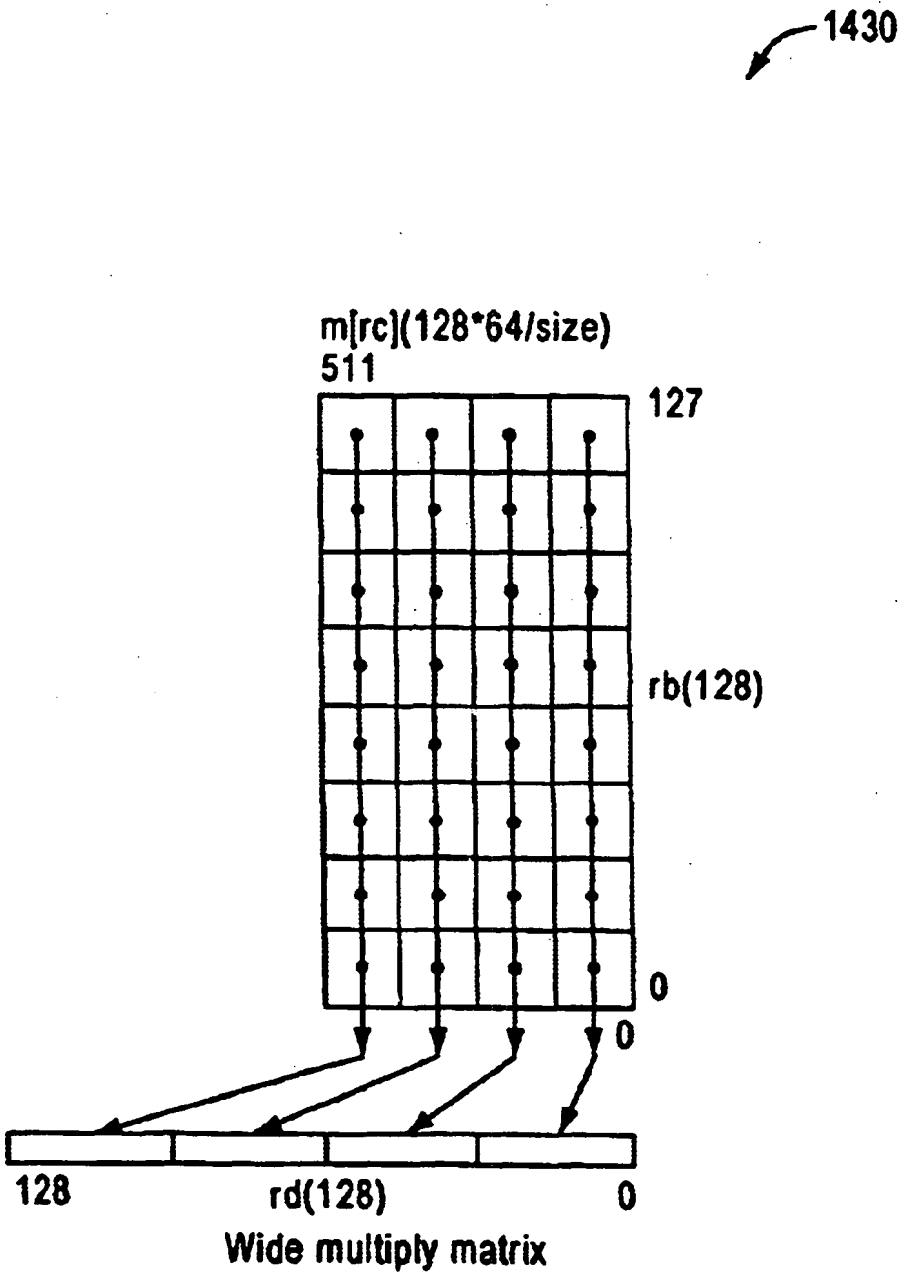

1430
m[rc](128*64/size)
511
127
rb(128)
0
0
128
rd(128)
0
Wide multiply matrix

FIG. 14B

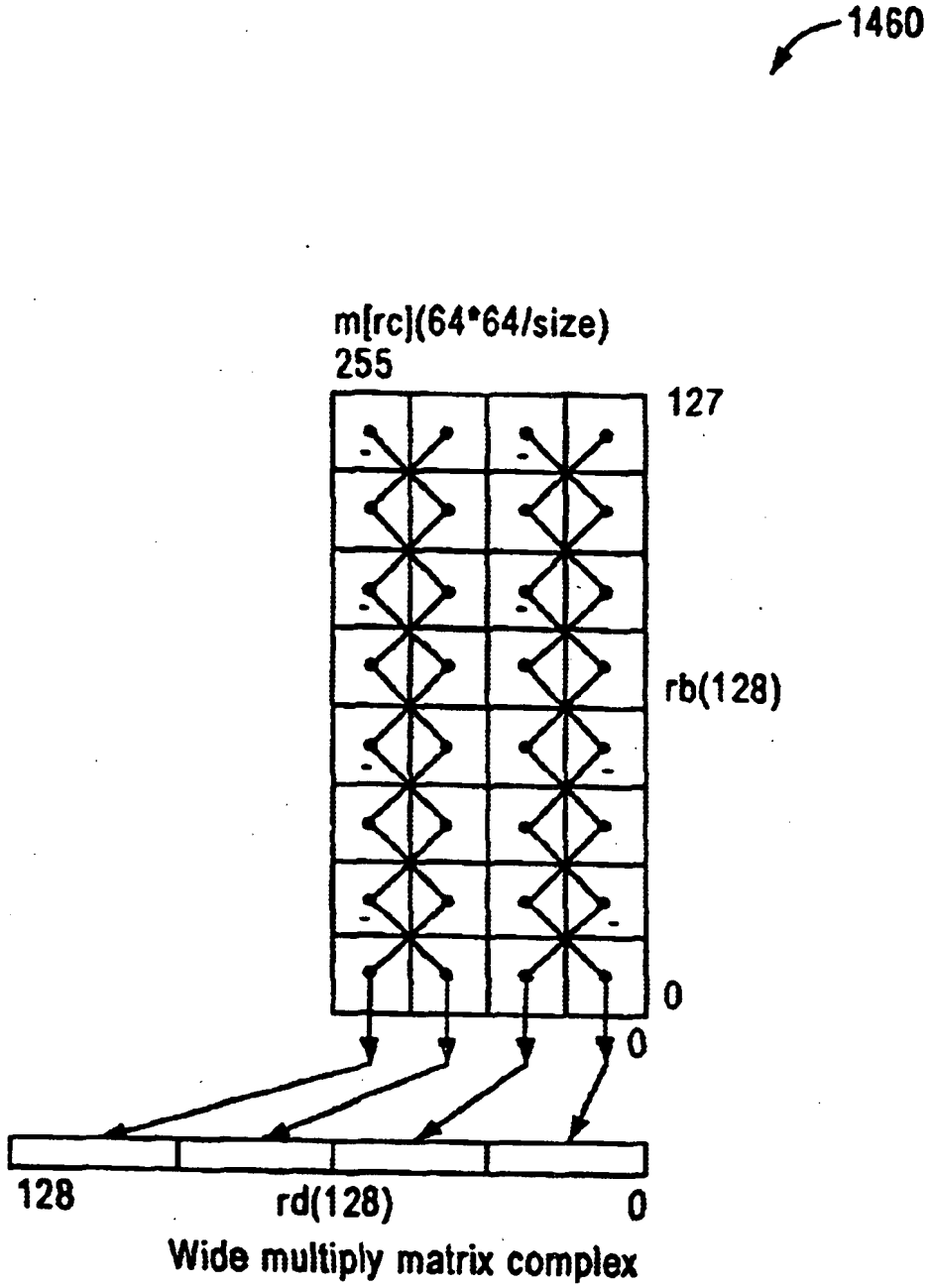

1460
m[rc](64*64/size)
255
127
rb(128)
0
0
128
rd(128)
0
Wide multiply matrix complex

FIG. 14C

1480

```
Definition
 def mul(size,h,vs,v,i,ws,j)as
     mul←((vs&v_{size-1+i})^{h-size} || v_{size-1+i..i}) °((ws&w_{size-1+j})^{h-size} || w_{size-1+j..j})
 enddef

 def c←PolyMultiply(size,a,b) as
     p[0]←0^{2*size}
     for k←0 to size-1
         p[k+1]←p[k] ^ a_k ? (0^{size-k} || b || 0^k) : 0^{2*size}
     endfor
     c←p[size]
 enddef

 def WideMultiplyMatrix(major,op,gsize,rd,rc,rb)
     d←RegRead(rd, 128)
     c←RegRead(rc, 64)
     b←RegRead(rb,128)
     lgsize←log(gsize)
     if c_{lgsize-4..0} ≠ 0 then
         raise AccessDisallowedByVirtualAddress
     endif
     if c_{2..lgsize-3} ≠ 0 then
         wsize←(c and (0-c)) || 0^4
         t←c and (c-1)
     else
         wsize←64
         t←a
     endif
     lwsize←log(wsize)
     if t_{lwsize+6-lgsize..lwsize-3} ≠ 0 then
         msize←(t and (0-t)) || 0^4
         VirtAddr←t and (t-1)
     else
         msize←128*wsize/gsize
         VirtAddr←t
     endif
     case major of
         W.MINOR.B:
             order←B
         W.MINOR.L:
             order←L
     endcase
```

FIG. 14D-1

1480

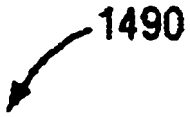
```
    case op of
        M.MUL.MAT.U.8, W.MUL.MAT.U.16, W.MUL.MAT.U.32,
        W.MUL.MAT.U.64:
            ms←bs←0
        W.MUL.MAT.M.8, W.MUL.MAT.M.16, W.MUL.MAT.M.32,
        W.MUL.MAT.M.64
            ms←0
            bs←1
        W.MUL.MAT.8, W.MUL.MAT.16, W.MUL.MAT.32,
        W.MUL.MAT.64, W.MUL.MAT.C.8, W.MUL.MAT.C.16,
        W.MUL.MAT.C.32, W.MUL.MAT.C.64:
            ms←bs←1
        W.MUL.MAT.P.8, W.MUL.MAT.P.16, W.MUL.MAT.P.32,
        W.MUL.MAT.P.64:
    endcase
m ← LoadMemory(c,VirtAddr,msize,order)
h ←2*gsize

for i ←0 to wsize-gsize by gsize
    q[0] ←0^(2*gsize)
    for j←0 to vsize-gsize by gsize
        case op of
            W.MUL.MAT.P.8, W.MUL.MAT.P.16,
            W.MUL.MAT.P.32, W.MUL.MAT.P.64:
                k ← i+wsize*j8..lgsize
                q[j+gsize] ← q[j] ^ PolyMultiply(gsize,m k+gsize-1..k,
                bj+gsize-1..j)
            W.MUL.MAT.C.8, W.MUL.MAT.C.16, W.MUL.MAT.C.32,
            W.MUL.MAT.C.64:
                if (~i) & gsize = 0 then
                    k←i-(j&gsize)+wsize*j8..lgsize+1
                    q[j+gsize]← q[i] + mul(gsize,h,ms,m,k,bs,b,j)
                else
                    k ← i+gsize+wsize*j8..lgsize+1
                    q[i+gsize]←q[i] = mul(gsize,h,ms,m,k,bs,b,j)
                endif
```

FIG. 14D-2

1480

```
                W.MUL.MAT.8, W.MUL.MAT.16, W.MUL.MAT.32,
                W.MUL.MAT.64, W.MUL.MAT.M.8, W.MUL.MAT.M.16,
                W.MUL.MAT.M.32, W.MUL.MAT.M.64, W.MUL.MAT.U.8,
                W.MUL.MAT.U.16, W.MUL.MAT.U.32, W.MUL.MAT.U.64
                    q[i+gsize] ← q[i] + mul(gsize,h,ms,m,i+wsize*
                    j8..lgsize,bs,b,i)
            endfor
            a2*gsize-1+2*i..2*i ← q[vsize]
        endfor
        a127..2*wsize ← 0
        RegWrite(rd, 128, a)
    enddef
```

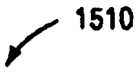
FIG. 14D-3



Exceptions

Access disallowed by virtual address
Access disallowed by tag
Access disallowed by global TB
Access disallowed by local TB
Access detail required by tag
Access detail required by local TB
Access detail required by global TB
Local TB miss
Global TB miss

FIG. 14E



Operation codes

W.MUL.MAT.X.B	Wide multiply matrix extract big-endian
W.MUL.MAT.X.L	Wide multiply matrix extract little-indian

Selection

class	op	order
Multiply matrix extract	W.MUL.MAT.X	B L

Format

W.op.order ra=rc,rd,rb

ra=wop(rc,rd,rb)

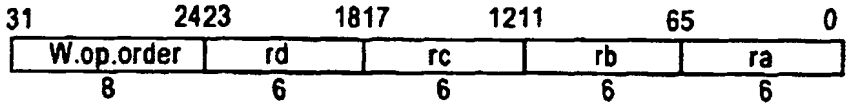


FIG. 15A

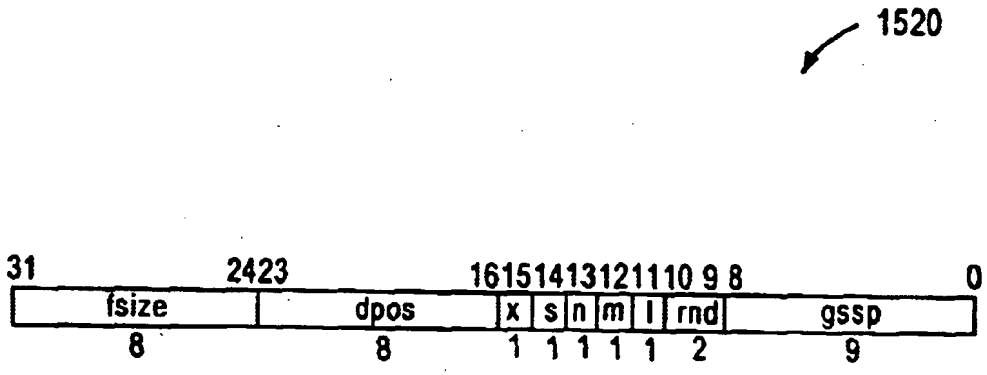

1520
31
2423
16151413121110 9 8
0
fsize
dpos
x
s
n
m
l
rnd
gssp
8
8
1 1 1 1 1 2
9

FIG. 15B

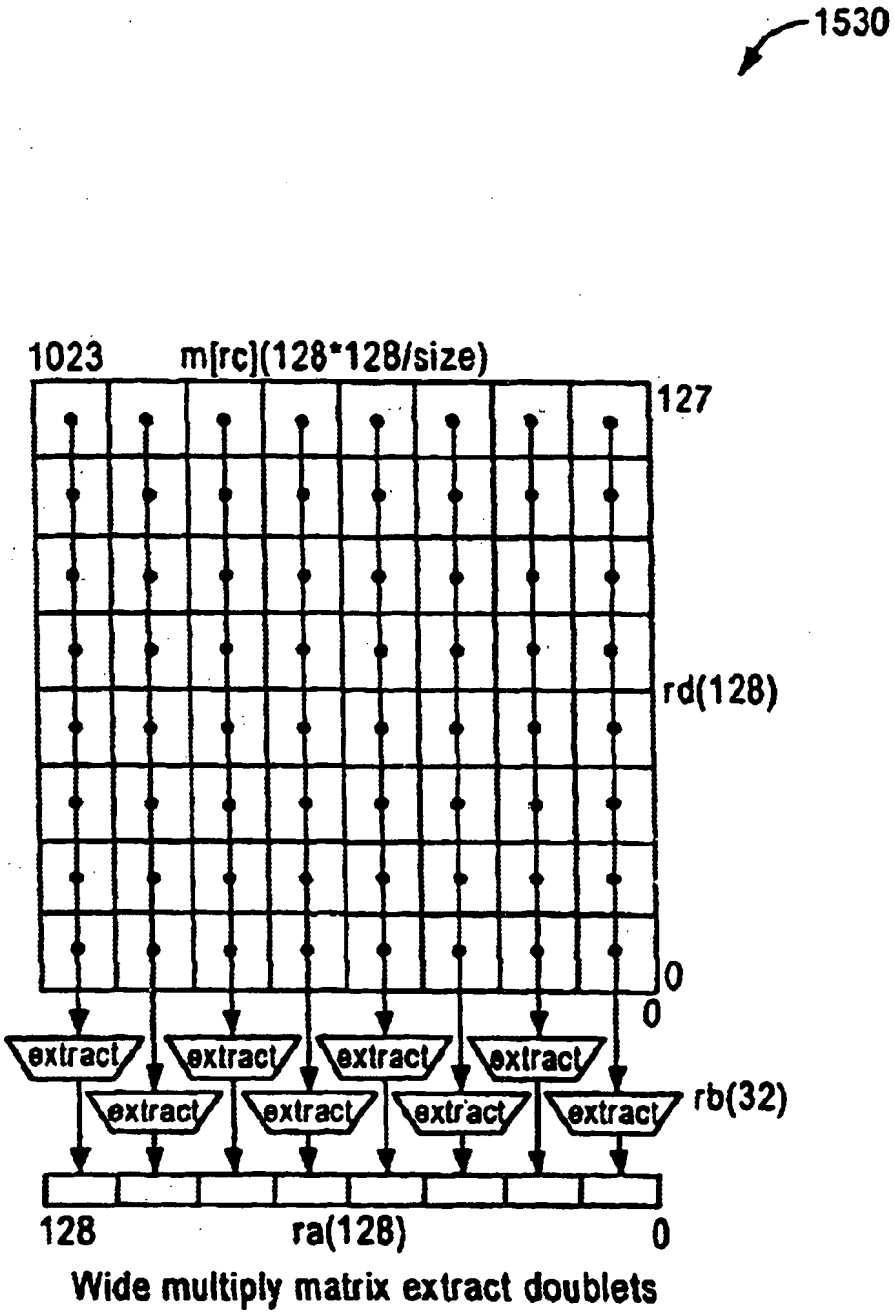

1530
1023
m[rc](128*128/size)
127
rd(128)
0
0
extract
extract
extract
extract
extract
extract
extract
extract
extract
rb(32)
128
ra(128)
0
Wide multiply matrix extract doublets

FIG. 15C

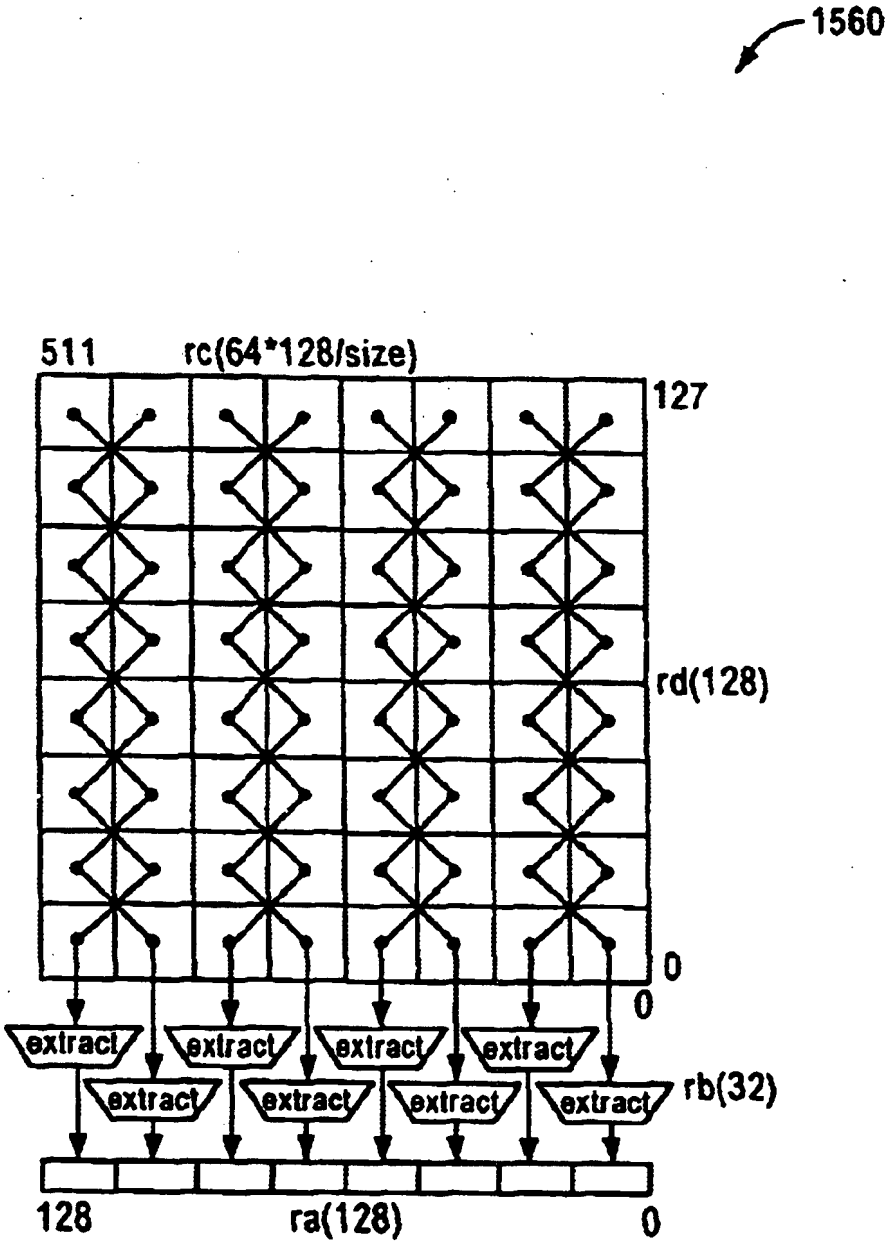

1560
511
rc(64*128/size)
127
rd(128)
0
0
extract
extract
extract
extract
extract
extract
extract
extract
extract
rb(32)
128
ra(128)
0
Wide multiply matrix extract complex doublets

FIG. 15D

```
Definition                                                    1580
def mul(size,h,vs,v,i,ws,w,j) as
    mul ← ((vs&v_size-1+i)^h-size||v_size-1+i..i) * ((ws&w_size-1+j)^h-size||w_size-1+j..j)
enddef

def WideMultiplyMatrixExtract(op,ra,rb,rc,rd)
    d←RegRead(rd, 128)
    c←RegRead(rc, 64)
    b←RegRead(rb, 128)
    case b8..0 of
        0..255:
            sgsize←128
        256..383:
            sgsize←64
        384..447:
            sgsize←32
        448..479:
            sgsize←16
        480..495:
            sgsize←8
        496..503:
            sgsize←4
        504..507:
            sgsize←2
        508..511:
            sgsize←1
    endcase
    l←b11
    m←b12
    n←b13
    signed←b14
    if c3..0 ≠ 0 then
        wsize←(c and (0-c))||0^4
        t←c and (c-1)
    else
        wsize←128
        t←c
    endif
    if sgsize < 8 then
        gsize←8
    elseif sgsize > wsize/2 then
        gsize←wsize/2
    else
```

FIG. 15E-1

1580

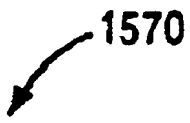
```
    gsize←sgsize
endif
lgsize←log(gsize
lwsize←log(wsize)
if t_{lwsize+6-n-lgsize..lwsize-3} ≠ 0 then
    msize ← (t and (0-t)) || 0^4
    VirtAddr ← t and (t-1)
else
    msize ← 64*(2-n)*wsize/gsize
    VirtAddr ← t
endif
vsize ← (1+n)*msize*gsize/wsize
mm ← LoadMemory(c,VirtAddr,msize,order)
lmsize ← log(msize)
if (VirtAddr_{lmsize-4..0} ≠ 0 then
    raise AccessDisallowedByVirtualAddress
endif
case op of
    W.MUL.MAT.X.B:
        order ← B
    W.MUL.MAT.X.L:
        order ← L
endcase
ms ← signed
ds ← signed ^ m
as ← signed or m
spos ← (b_{8..0}) and (2*gsize-1)
dpos ← (0 || b_{23..16}) and (gsize-1)
r ← spos
sfsize ← (0 || b_{31..24}) and (gsize-1)
tfsize ← (sfsize = 0) or ((sfsize+dpos) > gsize) ? gsize-dpos : sfsize
fsize ← (tfsize + spos > h) ? h - spos : tfsize
if (b_{10..9} = Z) & ~signed then
    rnd ← F
else
    rnd ← b_{10..9}
endif
```

FIG. 15E-2

1580

```
for i ← 0 to wsize-gsize by gsize
    q[0] ← 0^(2*gsize+7-lgsize)
    for j ← 0 to vsize-gsize by gsize
        if n then
            if (~) & j & gsize = 0 then
                k ← i-(j&gsize)+wsize*j_(8..lgsize+1)
                q[i+gsize] ← q[i] + mul(gsize,h,ms,mm,k,ds,d,j)
            else
                k ← i+gsize+wsize*j_(8..lgsize+1)
                q[i+gsize] ← q[i] - mul(gsize,h,ms,mm,k,ds,d,j)
            endif
        else
            q[i+gsize] ← q[i] = mul(gsize,h,ms,mm,i+j*wsize/gsize,ds,d,j)
        endif
    endfor
    p ← q[128]
    case rnd of
        none, N:
            s ← 0^(h-r) || ~p_r || p_(r-1)^(r-1)
        Z:
            s ← 0^(h-r) || p_(h-1)^r
        F:
            s ← 0^h
        C:
            s ← 0^(h-r) || 1^r
    endcase
    v ← ((ds & p^(h-1)) || p) + (0 || s)

        if (v_(h..r+fsize) = (as & v_(r+fsize-1))^(h+1-r-fsize)) or not I then
            w ← (as & v_(r+fsize-1))^(gsize-fsize-dpos) || v_(fsize-1+r..r) || 0^dpos
        else
            w ← (s ? (v_h || ~v_h^(gsize-dpos-1)) : 1^(gsize-dpos)) || 0^dpos
        endif
        a_(size-1+i..i) ← w
    endfor
    a_(127..wsize) ← 0
    RegWrite(ra, 128, a)
enddef
```

FIG. 15E-3



Exceptions

Access disallowed by virtual address
Access disallowed by tag
Access disallowed by global TB
Access disallowed by local TB
Access detail required by tag
Access detail required by local TB
Access detail required by global TB
Local TB miss
Global TB miss

FIG. 15F

1610

Operation codes

W.MUL.MAT.X.I.8.B	Wide multiply matrix extract immediate signed byte big-endian
W.MUL.MAT.X.I.8.L	Wide multiply matrix extract immediate signed byte little-endian
W.MUL.MAT.X.I.16.B	Wide multiply matrix extract immediate signed doublet big-endian
W.MUL.MAT.X.I.16.L	Wide multiply matrix extract immediate signed doublet little-endian
W.MUL.MAT.X.I.32.B	Wide multiply matrix extract immediate signed quadlet big-endian
W.MUL.MAT.X.I.32.L	Wide multiply matrix extract immediate signed quadlet little-endian
W.MUL.MAT.X.I.64.B	Wide multiply matrix extract immediate signed octlets big-endian
W.MUL.MAT.X.I.64.L	Wide multiply matrix extract immediate signed octlets little-endian
W.MUL.MAT.X.I.C.8.B	Wide multiply matrix extract immediate complex bytes big-endian
W.MUL.MAT.X.I.C.8.L	Wide multiply matrix extract immediate complex bytes little-endian
W.MUL.MAT.X.I.C.16.B	Wide multiply matrix extract immediate complex doublets big-endian
W.MUL.MAT.X.I.C.16.L	Wide multiply matrix extract immediate complex doublets little-endian
W.MUL.MAT.X.I.C.32.B	Wide multiply matrix extract immediate complex quadlets big-endian
W.MUL.MAT.X.I.C.32.L	Wide multiply matrix extract immediate complex quadlets little-endian

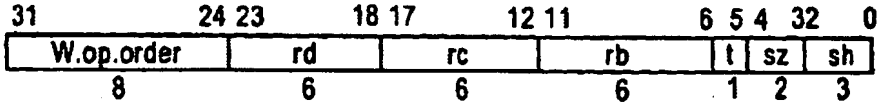
Selection

class	op	type	size	order
wide multiply extract immediate	W.MUL.MAT.X.I	NONE	8 16 32 64	L B
		C	8 16 32	L B

Format

W.op.tsize.order rd=rc,rb, i

rd=woptsizeorder(rc,rb,i)



sz ← log(size) - 3

assert size+3 ≥ i ≥ size-4

sh ← i - size

FIG. 16A

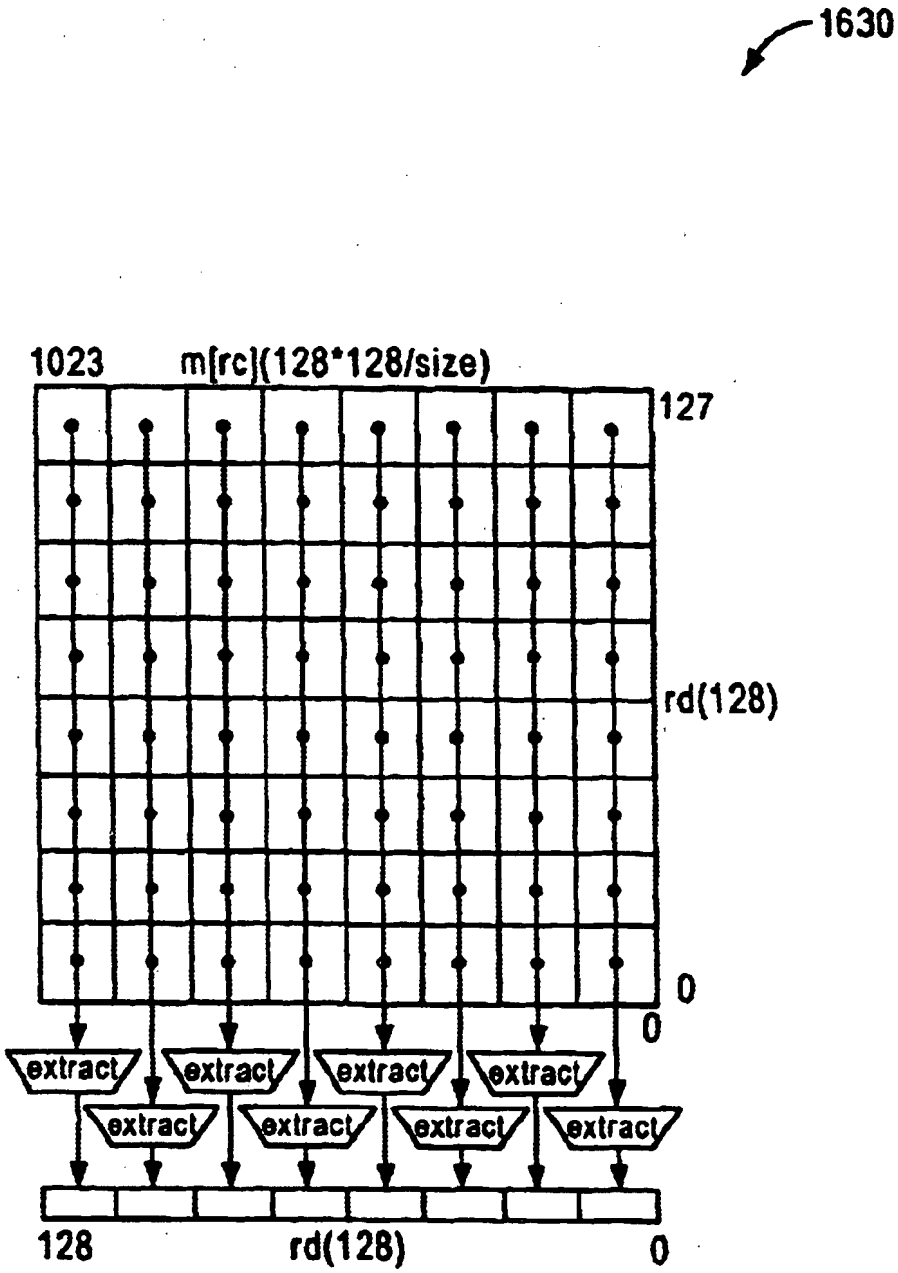
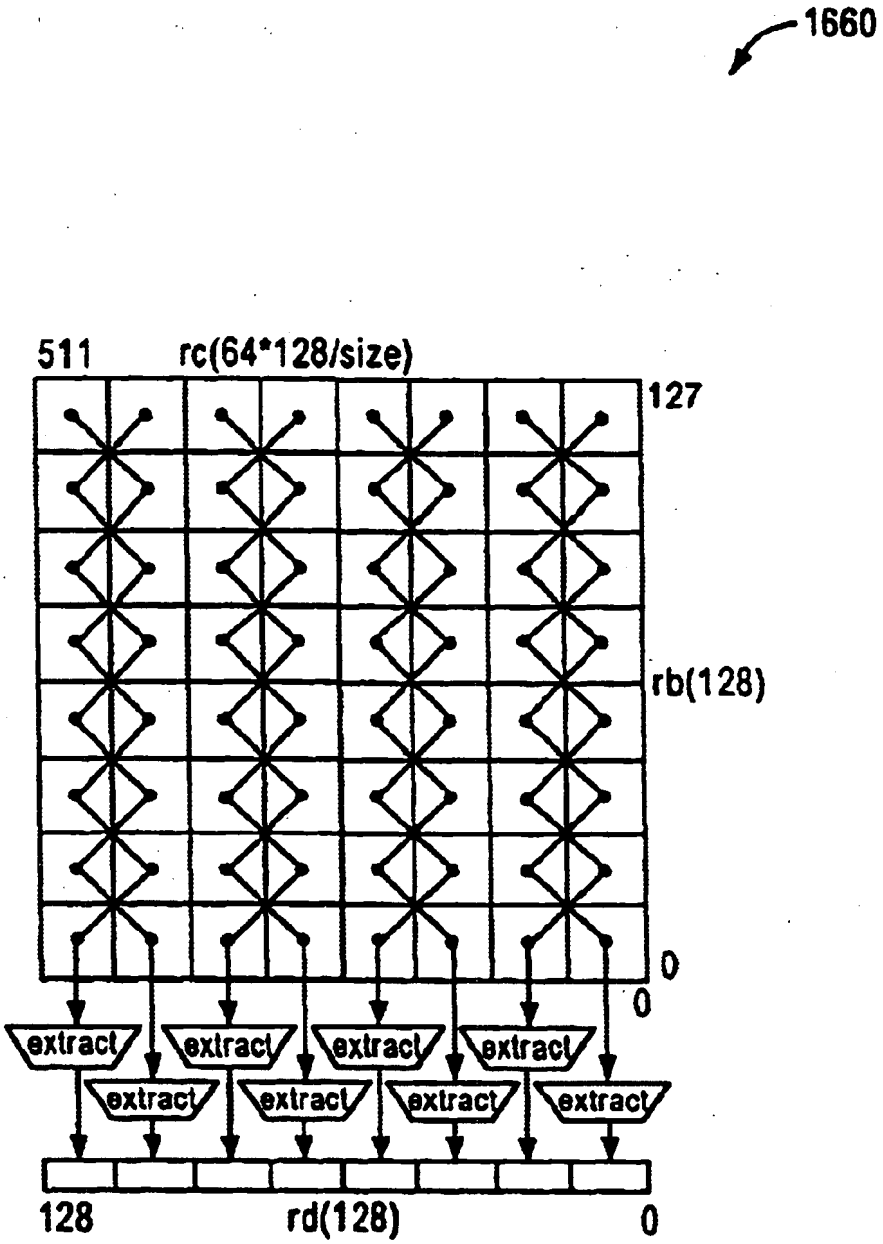

1630
1023
m[rc](128*128/size)
127
rd(128)
0
0
extract
extract
extract
extract
extract
extract
extract
extract
128
rd(128)
0
Wide multiply matrix extract immediate doublets

FIG. 16B



Wide multiply matrix extract immediate complex doublets

FIG. 16C

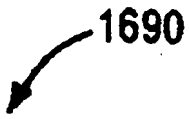
1680

```
Definition
def mul(size,h,vs,v,i,ws,w,j) as
    mul ←((vs&v_{size-1+i})^{h-size} || v_{size-1+i..i}) * ((ws&w_{size-1+j})^{h-size} || w_{size-1+j..j})
enddef

def WideMultiplyMatrixExtractimmediate(op,type,gsize,rd,rc,rb,sh)
    c ←RegRead(rc, 64)
    b ←RegRead(rb, 128)
    lgsize ←log(gsize)
    case type of
        NONE:
            if c_{lgsize-4..0} ≠ 0 then
                raise AccessDisallowedBy VirtualAddress
            endif
            if c_{3..lgsize-3} ≠ 0 then
                wsize ←(c and (0-c)) || 0^4
                t ←c and (c-1)
            else
                wsize ←128
                t ←c
            endif
            lwsize ←log(wsize)
            if t_{lwsize+6-lgsize..lwsize-3} ≠ 0 then
                msize ←(t and (0-t)) || 0^4
                VirtAddr ←t and (t-1)
            else
                msize ←128*wsize/gsize
                VirtAddr ←t
        C:
            if c_{lgsize-4..0} ≠ 0 then
                raise AccessDisallowedByVirtualAddress
            endif
            if c_{3..lgsize-3} ≠ 0 then
                wsize ←(c and (0-c)) || 0^4
                t ←c and (c-1)
            else
                wsize ←128
                t ←c
            endif
            lwsize ←log(wsize)
            if t_{lwsize+5-lgsize..lwsize-3} ≠ 0 then
                msize ←(t and (0-t)) || 0^4
```

FIG. 16D-1

```
                    VirtAddr ← t and (t-1)                    ←1680
                else
                    msize ← 64*wsize/gsize
                    VirtAddr ← t
                endif
                vsize ← 2*msize*gsize/wsize
        endcase
        case of of
            W.MUL.MAT.X.I.B:
                order ← B
            W.MUL.MAT.X.I.L:
                order ← L
        endcase
        as ← ms ← bs ← 1
        m ← LoadMemory(c,VirtAddr,msize,order)
        h ← (2*gsize) + 7 - lgsize-(ms and bs)
        r ← gsize + (sh[5]2||sh)
        for ← 0 to wsize-gsize by gsize
            q[0] ← 0^(2*gsize+7-lgsize)
            for j ← 0 to vsize-gsize by gsize
                case type of
                    NONE:
                        q[j+gsize] ← q[i] + mul(gsize,h,ms,m,i+wsize*
                        j[8..lgsize],bs,b,j)
                    C:
                        if (~i) & j & gsize = 0 then
                            k ← i-(j&gsize)+wsize*j[8..lgsize+1]
                            q[j+gsize] ← q[i] + mul(gsize,h,ms,m,k,bs,b,j)
                        else
                            k ← i+gsize+wsize*j[8..lgsize+1]
                            q[j+gsize] ← q[j] - mul(gsize,h,ms,m,k,bs,b,j)
                        endif
                endcase
            endfor
            p ← q[vsize]
            s ← 0^(h-r) || ~p_r || p_r^(r-1)
            v ← ((as & p_(h-1)) || p) + (0||s)
            if (v[h..r+gsize] = (as & v[r+gsize-1])^(h+1-r-gsize) then
                a[gsize-1+i..i] ← v[gsize-1+r..r]
            else
                a[gsize-1+i..i] ← as ? (v_h || ~v_h^(gsize-1)) : 1gsize
            endif
        endfor
        a[127..wsize] ← 0
        RegWrite(rd, 128, a)
enddef
```

FIG. 16D-2



Exceptions

Access disallowed by virtual address
Access disallowed by tag
Access disallowed by global TB
Access disallowed by local TB
Access detail required by tag
Access detail required by local TB
Access detail required by global TB
Local TB miss
Global TB miss

FIG. 16E

1710

Operation codes

W.MUL.MAT.C.F.16.B	Wide multiply matrix complex floating-point half big-endian
W.MUL.MAT.C.F.16.L	Wide multiply matrix complex floating-point little-endian
W.MUL.MAT.C.F.32.B	Wide multiply matrix complex floating-point single big-endian
W.MUL.MAT.C.F.32.L	Wide multiply matrix complex floating-point single little-endian
W.MUL.MAT.F.16.B	Wide multiply matrix floating-point half big-endian
W.MUL.MAT.F.16.L	Wide multiply matrix floating-point half little-endian
W.MUL.MAT.F.32.B	Wide multiply matrix floating-point single big-endian
W.MUL.MAT.F.32.L	Wide multiply matrix floating-point single little-endian
W.MUL.MAT.F.64.B	Wide multiply matrix floating-point double big-endian
W.MUL.MAT.F.64.L	Wide multiply matrix floating-point double little-endian

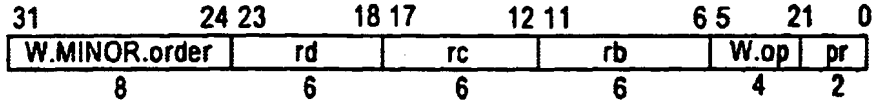
Selection

class	op	type	prec	order
wide multiply matrix	W.MUL.MAT	F	16 32 64	L B
		C.F	16 32	L B

Format

W.op.prec.order rd=rc,rb

rd=wopprecorder(rc,rb)



Pr ← log(prec) - 3

FIG. 17A

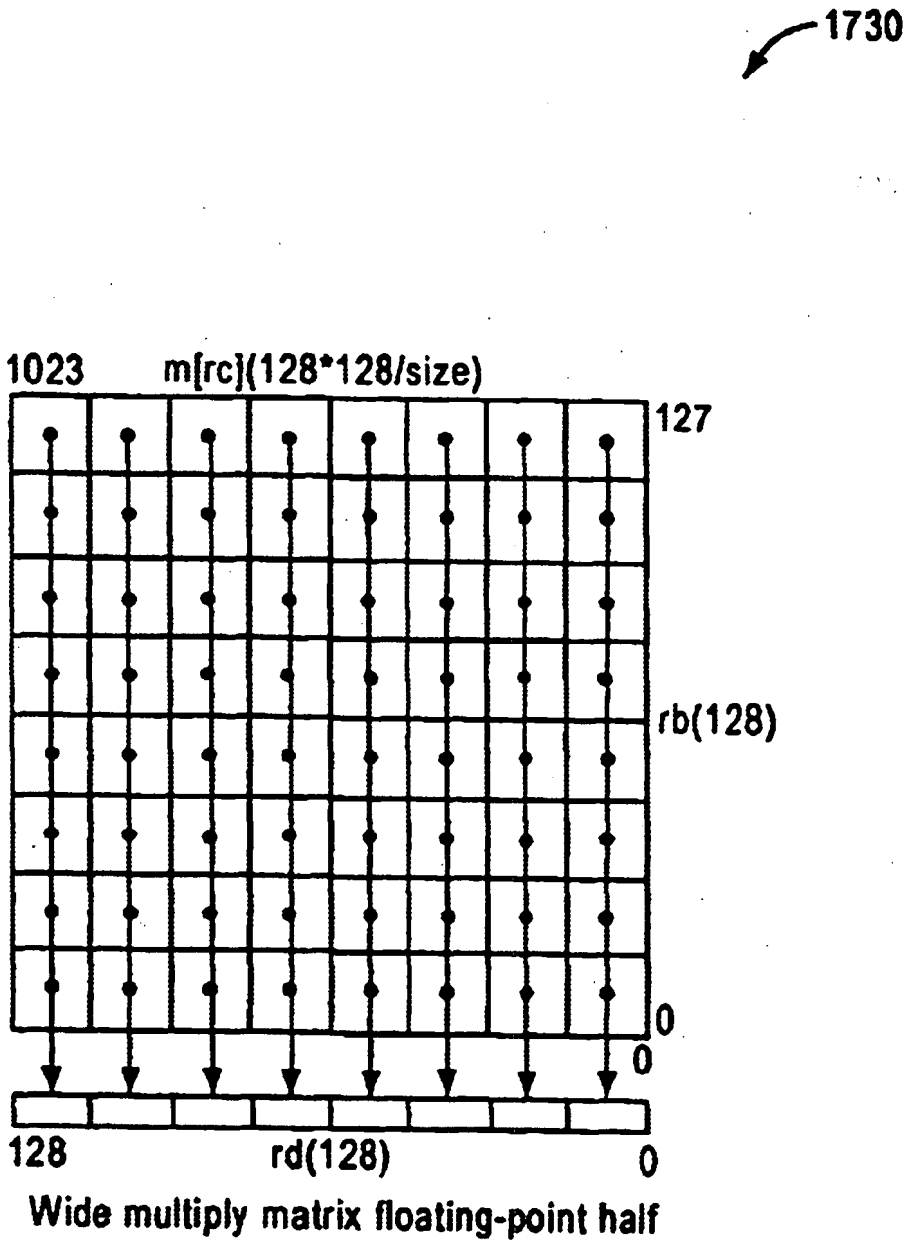

1730
1023
m[rc](128*128/size)
127
rb(128)
0
0
128
rd(128)
0
Wide multiply matrix floating-point half

FIG. 17B

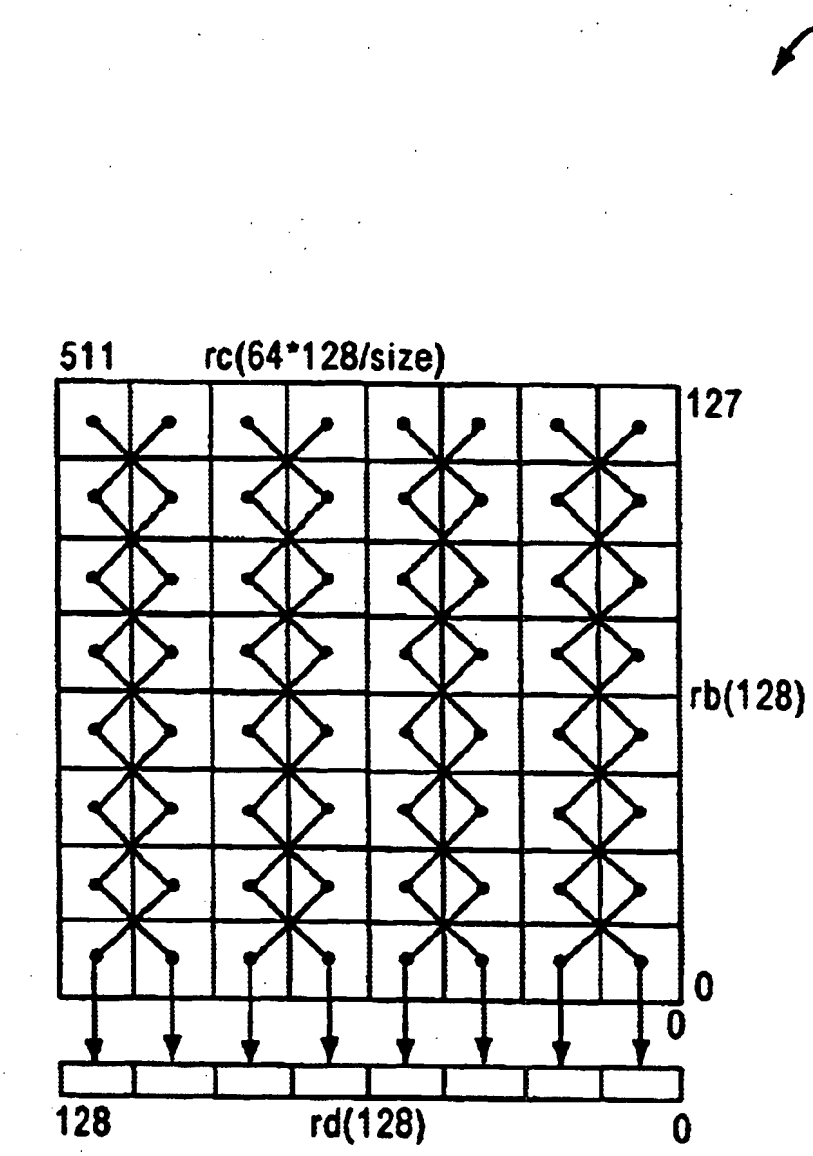

1760
511
rc(64*128/size)
127
rb(128)
0
0
128
rd(128)
0
Wide multiply matrix complex floating-point half

FIG. 17C

1780

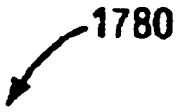
```
Definition
def mul(size,v,i,w,j) as
    mul ← fmul(F(size,v_{size-1+i..i}),F(size,w_{size-1+j..j}))
enddef

def WideMultiplyMatrixFloatingPoint(major,op,gsize,rd,rc,rb)
    c ← RegRead(rc, 64)
    b ← RegRead(rb, 128)
    lgsize ← log(gsize)
    switch op of
        W.MUL.MAT.F.16, W.MUL.MAT.F.32, W.MUL.MAT.F.64:
            if c_{lgsize-4..0} ≠ 0 then
                raise AccessDisallowedByVirtualAddress
            endif
            if c_{3..lgsize-3} ≠ 0 then
                wsize ← (c and (0-c))|| 0^4
                t ← c and (c-1)
            else
                wsize ← 128
                t ← c
            endif
            lwsize ← log(wsize)
            if t_{lwsize+6-lgsize..lwsize-3} ≠ 0 then
                msize ← (t and (0-t))|| 0^4
                VirtAddr ← t and (t-1)
            else
                msize ← 128*wsize/gsize
                VirtAddr ← t
            endif
            vsize ← msize*gsize/wsize
        W.MUL.MAT.C.F.16, W.MUL.MAT.C.F.32, W.MUL.MAT.C.F.64:
            if c_{lgsize-4..0} ≠ 0 then
                raise AccessDisallowedByVirtualAddress
            endif
            if c_{3..lgsize-3} ≠ 0 then
                wsize ← (c and (0-c))|| 0^4
                t ← c and (c-1)
            else
                wsize ← 128
                t ← c
            endif
            lwsize ← log(wsize)
            if t_{lwsize+5-lgsize..lwsize-3} ≠ 0 then
```

FIG. 17D-1

1780

```
                msize←(t and (0-t))|| 0^4
                VirtAddr←t and (t-1)
            else
                msize←64*wsize/gsize
                VirtAddr←t
            endif
            vsize←2*msize*gsize/wsize
        endcase
        case major of
            M.MINOR.B:
                order←B
            M.MINOR.L:
                order←L
        endcase
        m←LoadMemory(c,VirtAddr,msize,order)
        for i←0 to wsize-gsize by gsize
            q[0].t←NULL
            for j←0 to vsize-gsize by gsize
                case op of
                    W.MUL.MAT.F.16, W.MUL.MAT.F.32, W.MUL.MAT.F.64:
                        q[j+gsize]←faddq[j], mul(gsize,m,i+wsize*
                        j8..lgsize+1,b,j))
                    W.MUL.MAT.C.F.16, W.MUL.MAT.C.F.32,
                    W.MUL.MAT.C.F.64:
                        if (~i) & j & gsize = 0 then
                            k←i-(j&gsize)+wsize*j8..lgsize+1
                            q[j+gsize]←faqq[j], mul(gsize,m,k,b,j))
                        else
                            k←i+gsize+wsize*j8..lgsize+1
                            q[j+gsize]←fsubq[j], mul(gsize,m,k,b,j))
                        endif
                endcase
            endfor
            agsize-1+i..i←q[vsize]
        endfor
        a127..wsize←0
        RegWrite(rd, 128, a)
enddef
```

FIG. 17D-2



Exceptions

Floating-point arithmetic
Access disallowed by virtual address
Access disallowed by tag
Access disallowed by global TB
Access disallowed by local TB
Access detail required by tag
Access detail required by local TB
Access detail required by global TB
Local TB miss
Global TB miss

FIG. 17E

1810

Operation codes

W.MUL.MAT.G.8.B	Wide multiply matrix Galois bytes big-endian
W.MUL.MAT.G.8.L	Wide multiply matrix Galois bytes little-endian

Selection

class	op	size	order
Multiply matrix Galois	W.MUL.MAT.G	8	B L

Format

W.op.order ra=rc,rd,rb

ra=woporder(rc,rd,rb)

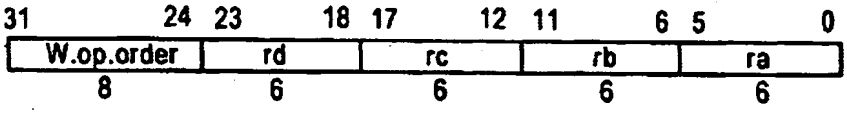


FIG. 18A

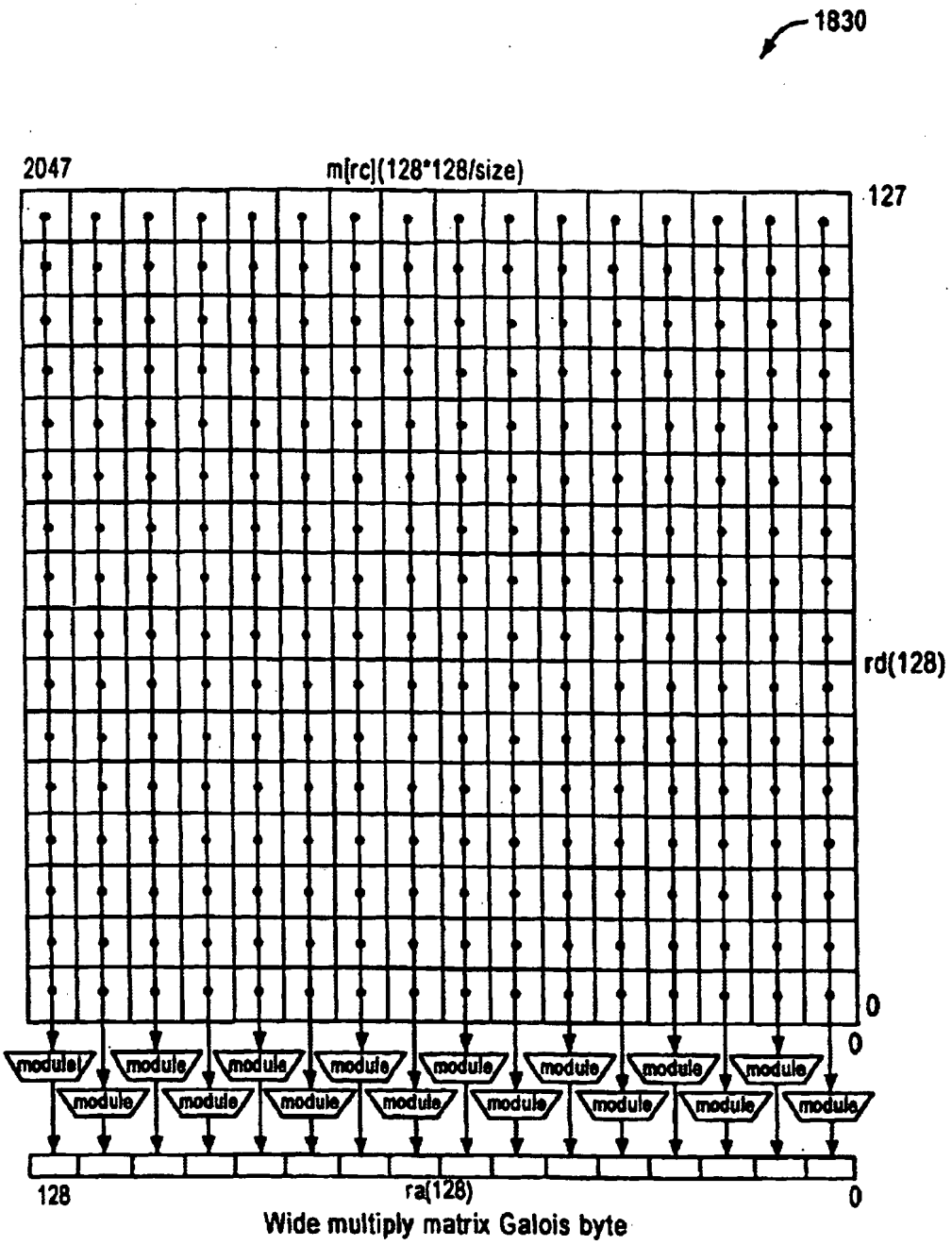

1830
2047
m[rc](128*128/size)
127
rd(128)
0
0
module
128
ra(128)
0
Wide multiply matrix Galois byte

FIG. 18B

1860

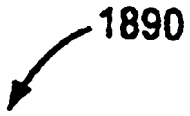
```
Definition
  def c←PolyMultiply(size,a,b) as
      p[0]←0^(2*size)
      for k←0 to size-1
          p[k+1]←p[k] ^ a_k ? (0^(size-k) || b || 0^k) : 0^(2*size)
      endfor
      c←p[size]
  enddef

  def c←PolyResidue(size,a,b) as
      p[0] ←a
      for k←size-1 to 0 by-1
          p[k-1]←p[k] ^ p[0]_(size+k) ?(0^(size-k) || 1^1 || b || 0^k) : 0^(2*size)
      endfor
      c←p[size]_(size-1..0)
  enddef
  def WideMultiplyMatrixGalois(op,gsize,rd,rc,rb,ra)
      d←RegRead(rd, 128)
      c←RegRead(rc, 64)
      b←RegRead(rb,128)
      lgsize←log(gsize)
      if c_(lgsize-4..0) ≠ 0 then
          raise AccessDisallowedByVirtualAddress
      endif
      if c_(3..lgsize-3) ≠ 0 then
          wsize←(c and (0-c)) || 0^4
          t←c and (c-1)
      else
          wsize←128
          t←c
      endif
      lwsize←log(wsize)
      if t_(lwsize+6-lgsize..lwsize-3) ≠ 0 then
          msize←(t and (0-t)) || 0^4
          VirtAddr←t and (t-1)
      else
          msize←128*wsize/gsize
          VirtAddr←t
      endif
      case op of
          W.MUL.MAT.G.8.B:
              order←B
          W.MUL.MAT.G.8.L:
              order←L
      endcase
```

FIG. 18C-1

1860

```
m ← LoadMemory(c, VirtAddr,msize,order)
for i ← 0 wsize-gsize by gsize
    q[0] ← 0^(2*gsize)
    for j ← 0 to vsize-gsize by gsize
        k ← i+wsize*j_(8..lgsize)
        q[j+gsize] ← q[j] ^ PolyMultiply(gsize, m_(k+gsize-1..k) , d_(j+gsize-1..j) )
    endfor

    a_(gsize-1+i..i) ← PolyResidue(gsize,q[vsize],b_(gsize-1..0) )
endfor
a_(127..wsize) ← 0
RegWrite(ra,128, a)
enddef
```

FIG. 18C-2



Exceptions

Access disallowed by virtual address
Access disallowed by tag
Access disallowed by global TB
Access disallowed by local TB
Access detail required by tag
Access detail required by local TB
Access detail required by global TB
Local TB miss
Global TB miss

FIG. 18D

1910

Operation codes

E.MUL.ADD.X	Ensemble multiply add extract
E.CON.X	Ensemble convolve extract

Format

E.op rd@rc,rb,ra

rd=gop(rd,rc,rb,ra)

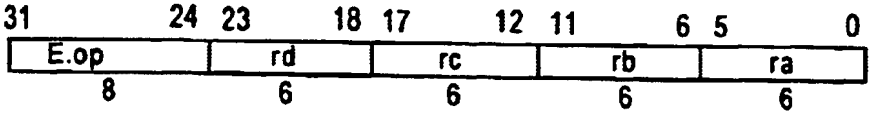


FIG. 19A

1910

Figures 19B and 20B has blank fields: should be.

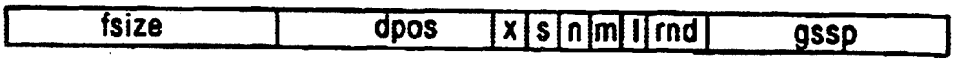


FIG. 19B

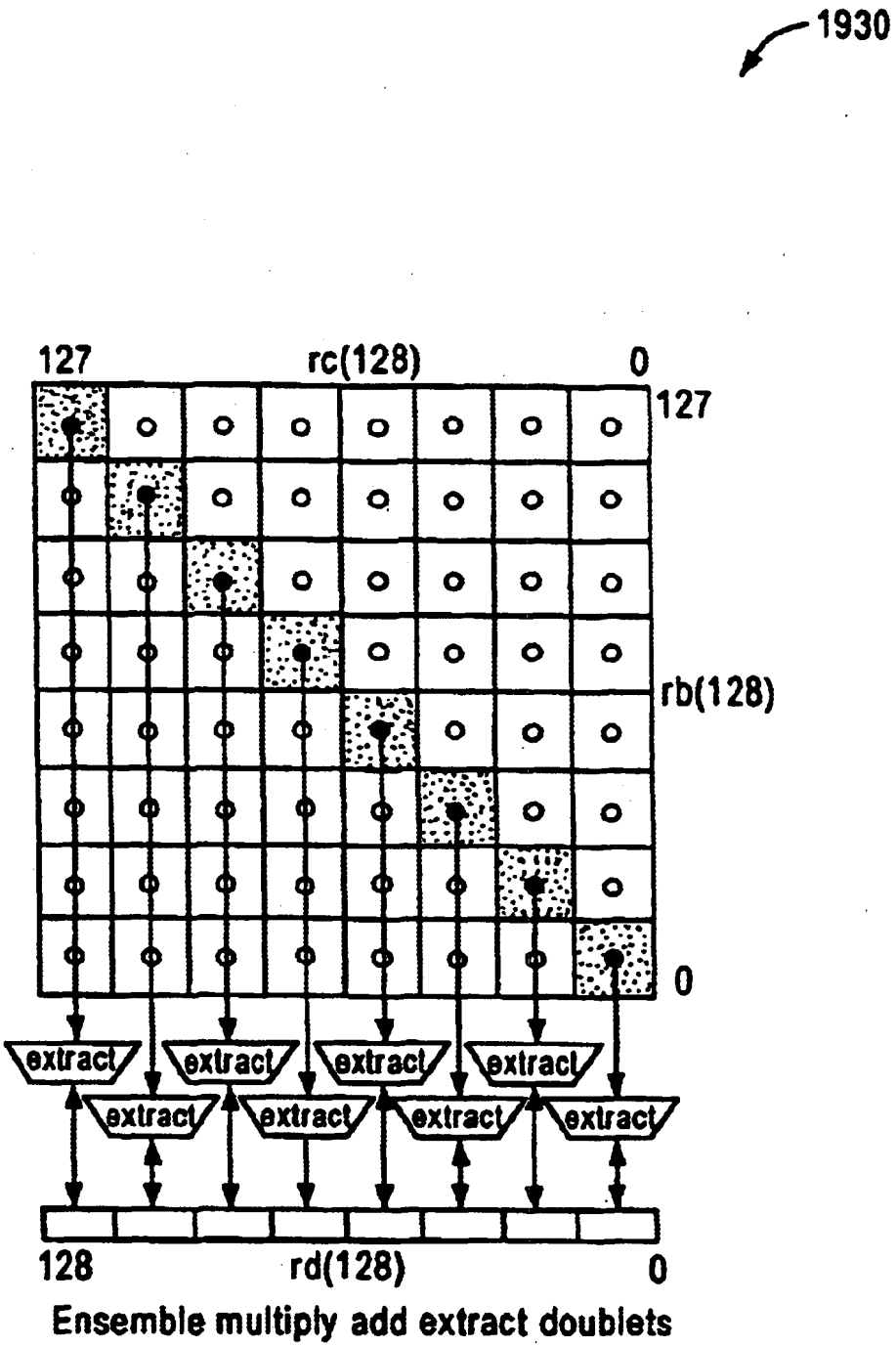
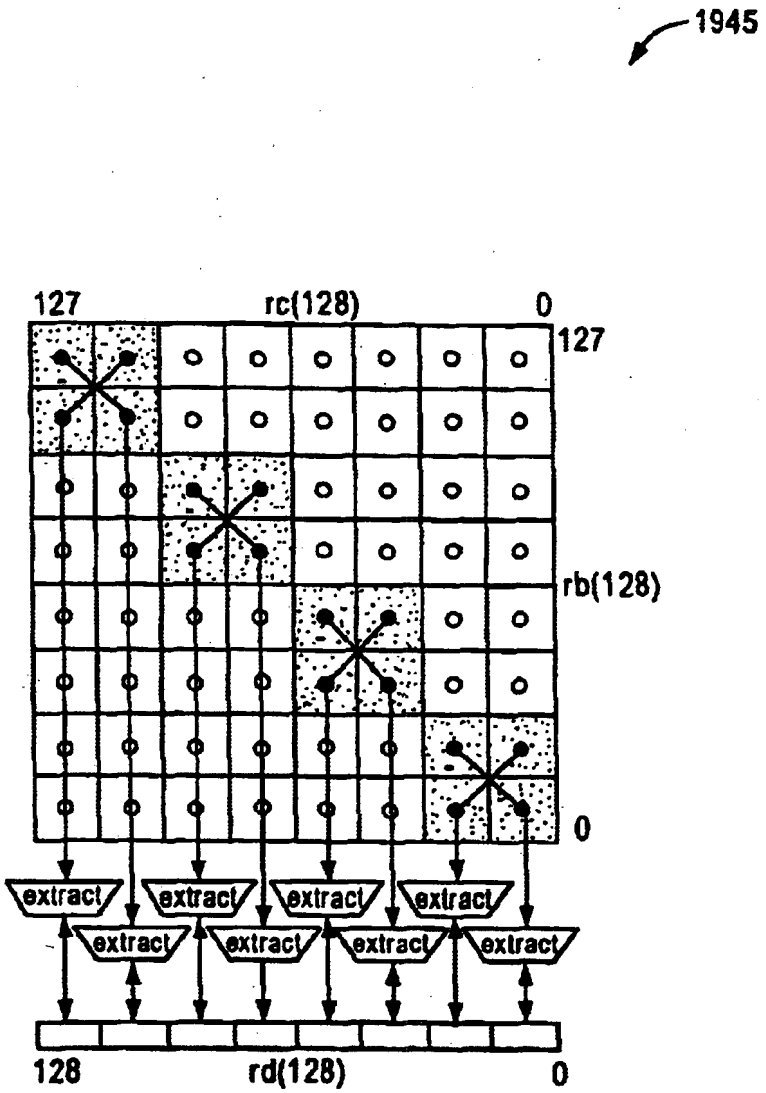

1930
127
rc(128)
0
127
rb(128)
0
extract
extract
extract
extract
extract
extract
extract
extract
extract
128
rd(128)
0
Ensemble multiply add extract doublets

FIG. 19C



Ensemble complex multiply add extract doublets

This ensemble-multiply-add-extract instructions (E.MUL.ADD.X), when the x bit is set, multiply the low-order 64 bits of each of the rc and rb registers and produce extended (double-size) results.

FIG. 19D

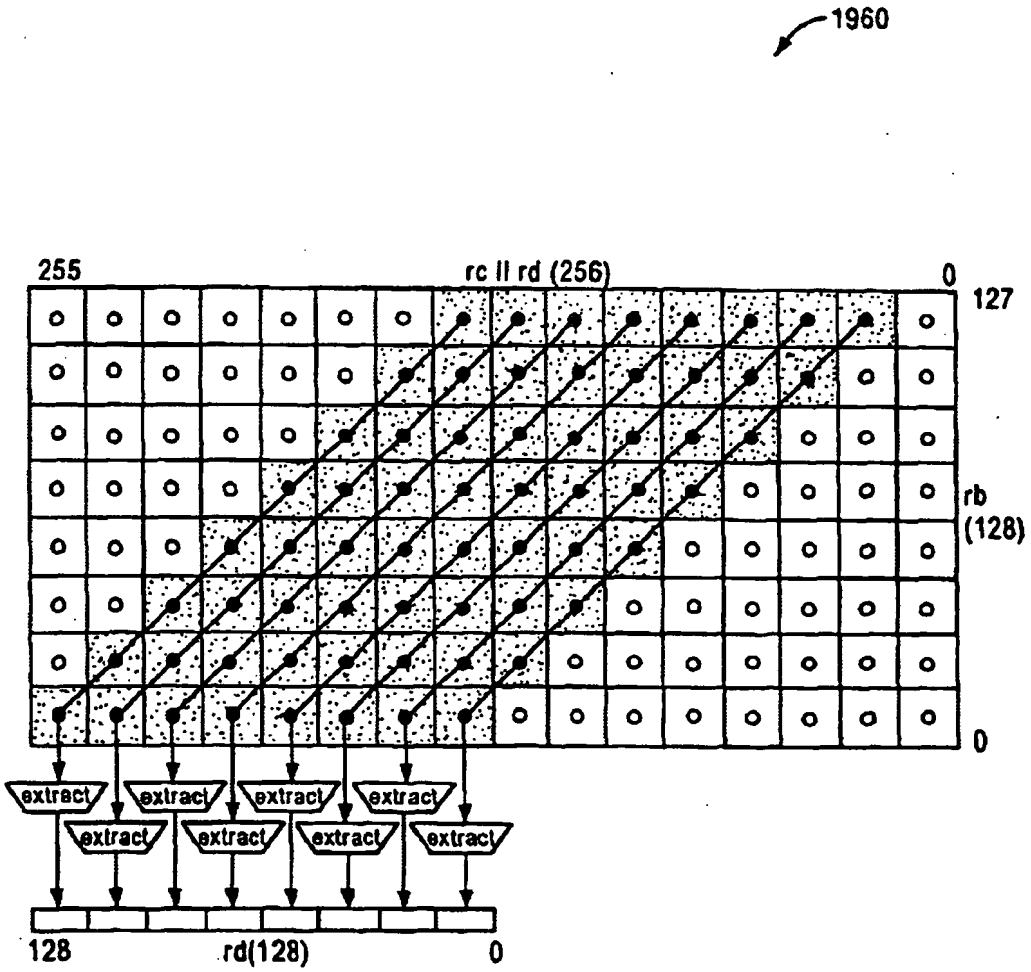

1960
255
rc II rd (256)
0
127
rb
(128)
0
extract
extract
extract
extract
extract
extract
extract
extract
extract
128
rd(128)
0
Ensemble convolve extract doublets

FIG. 19E

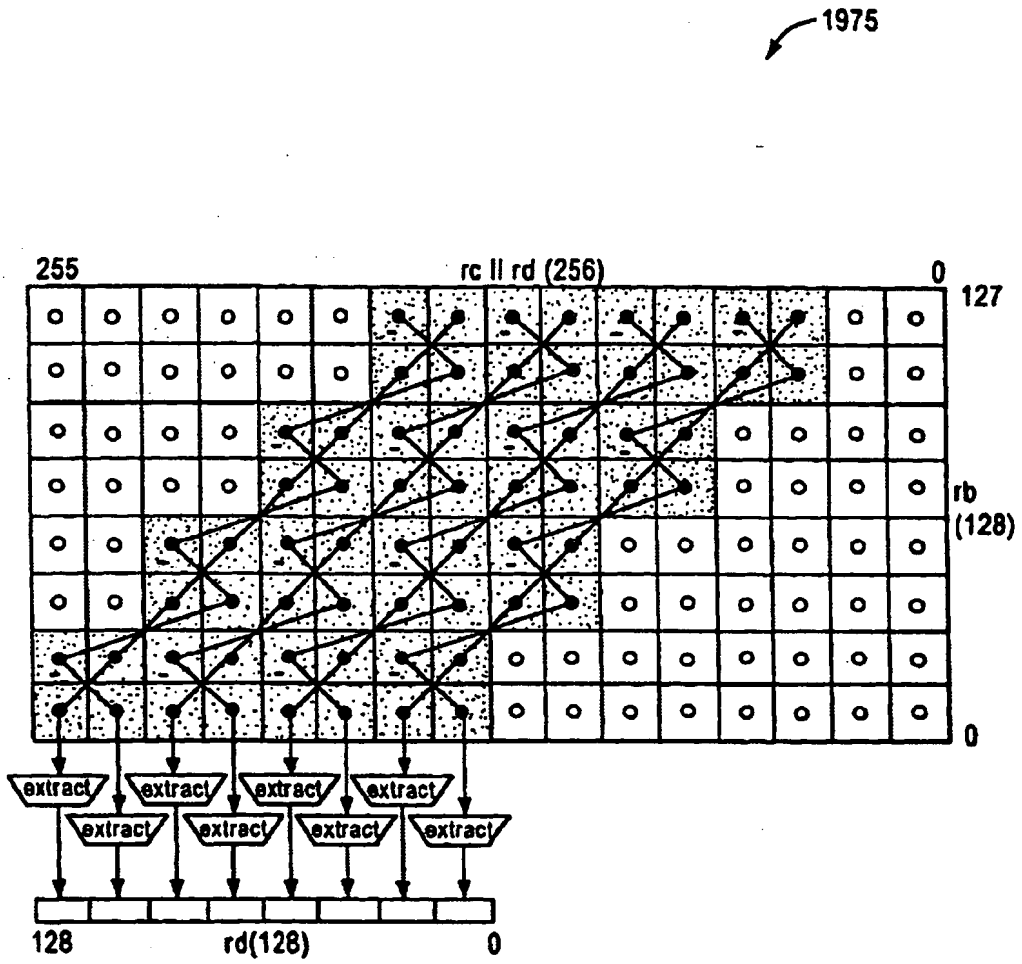

1975
255
rc II rd (256)
0
127
rb
(128)
0
extract
extract
extract
extract
extract
extract
extract
extract
extract
128
rd(128)
0
Ensemble convolve extract complex doublets

FIG. 19F

```
Definition                                                          1990
def mul(size,h,vs,v,i,ws,w,j) as
    mul← ((vs&v size-1+i)h-size||v size-1+i..i) * ((ws&w size-1+j)h-size||w size-1+j..j)
enddef

def EnsembleExtractInplace(op,ra,rb,rc,rd) as
    d←RegRead(rd, 128)
    c←RegRead(rc, 128)
    b←RegRead(rb, 128)
    case b8..0 of
        0..255:
            sgsize←128
        256..383:
            sgsize←64
        384..447:
            sgsize←32
        448..479:
            sgsize←16
        480..495:
            sgsize←8
        496..503:
            sgsize←4
        504..507:
            sgsize←2
        508..511:
            sgsize←1
    endcase
    l←a11
    m←a12
    n←a13
    signed←a14
    x←a15
    case op of
        E.CON.X:
            if (sgsize < 8) then
                gsize← 8
            elseif (sgsize*(n-1)*(x+1) > 128 then
                gsize← 128/(n-1)/(x+1)
            else
                gsize← sgsize
            endif
            lgsize← log(gsize)
            wsize ← 128/(x+1)
```

FIG. 19G-1

1990

```
        vsize ← 128
        ds ← cs ← signed
        bs ← signed ^ m
        zs ← signed or m or n
        zsize ← gsize*(x+1)
        h ← (2*gsize) + log(vsize) - lgsize
        spos ← (a8..0) and (2*gsize-1)

E.MUL.ADD.X:
    if(sgsize < 9) then
        gsize ← 8
    elseif (sgsize*(n+1)*(x+1) > 128) then
        gsize ← 128/(n+1)/(x+1)
    else
        gsize ← sgsize
    endif
    ds ← signed
    cs ← signed ^ m
    zs ← signed or m or n
    zsize ← gsize*(x+1)
    h ← (2*gsize) + n
    spos ← (a8..0) and (2*gssize-1)
endcase
dpos ← (0|| a23..16) and (zsize-1)
r ← spos
sfsize ← (0|| a31..24) and (zsize-1)
tfsize ← (sfsize = 0) or ((sfsize+dpos) > zsize) ? zsize-dpos : sfsize
fsize ← (tfsize + spos > h) ? h - spos : tfsize
if (b10..9 = Z) and not as then
    rnd ← F
else
    rnd ← b10..9
endif
```

FIG. 19G-2

1990

```
for k ← 0 to wsize-zsize by zsize
      i ← k*gsize/zsize
      case op of
          E.CON.X:
              q[0] ← 0
               for j ← 0 to vsize-gsize by gsize
                   if n then
                       if(~) & j &  gsize = 0 then
                           q[j+gsize] ← q[j] + mul(gsize,h,ms,m,i+
                           128-j,bs,b,j)
                        else
                            q[j+gsize] ← q[j] - mul(gsize,h,ms,i+
                            128-j+2*gsize,bs,b,j)
                         endif
                    else
                          q[j+gsize] ← q[j] + mul(gsize,h,ms,m,i+
                          128-j,bs,b,j)
                     endif
                endfor
                p ← q[vsize]
           E.MUL.ADD.X:
                di ← ((ds and dk+zize-1)h-zsize-r || (dk+zsize-1..k )|| 0r)
                if n then
                     if ( i and gsize) = 0 then
                          p ← mul(gsize,h,ds,d,i,cs,c,i)-
mul(gsize,h,ds,d,i+gsize,cs,c,i+gsize)+di
                     else

   p ← mul(gsize,h,ds,d,i,cs,c,i+gsize)+mul(gsize,h,ds,d,i,cs;c,i+gsize)+di
                     endif
                else
                      p ← mul(gsize,h,ds,d,i,cs,c,i) + di
                endif
      endcase
```

FIG. 19G-3

1990

```
            case rnd of
                N:
                    s ← 0^(h-r) || ~p_r || p_r^(r-1)
                Z:
                    s ← 0^(h-r) || p_(h-1)^r
                F:
                    s ← 0^h
                C:
                    s ← 0^(h-r) || 1^r
            endcase
            v ← ((zs & p_(h-1)) || p) + (0 || s)
            if (v_(h..r+fsize) = (zs & v_(r+fsize-1))^(h+1-r-fsize)) or not (I and (op =
            EXTRACT)) then
                w ← (zs & v_(r+fsize-1))^(zsize-fsize-dpos) || v_(fsize-1+r..r) || 0^dpos
            else
                w ← (zs ? (v_h || ~v_h^(zsize-dpos-1)) : 1^(zsize-dpos)) || 0^dpos
            endif
            Zzsize-1_k..k ← w
        endfor
        RegWrite(rd, 128, z)
    enddef
```

FIG. 19G-4

2010

Operation codes

E.MUL.X	Ensemble multiply extract
E.EXTRACT	Ensemble extract
E.SCAL.ADD.X	Ensemble scale and extract

Format

E.op ra=rd,rc,rb

ra=eop(rd,rc,rb)

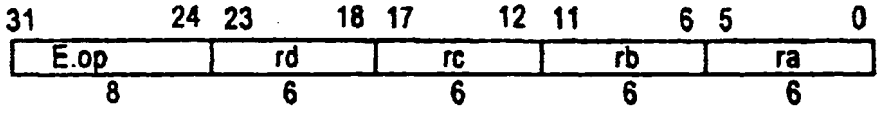


FIG. 20A

2015

Figures 19B and 20B has blank fields: should be.

FIG. 20B

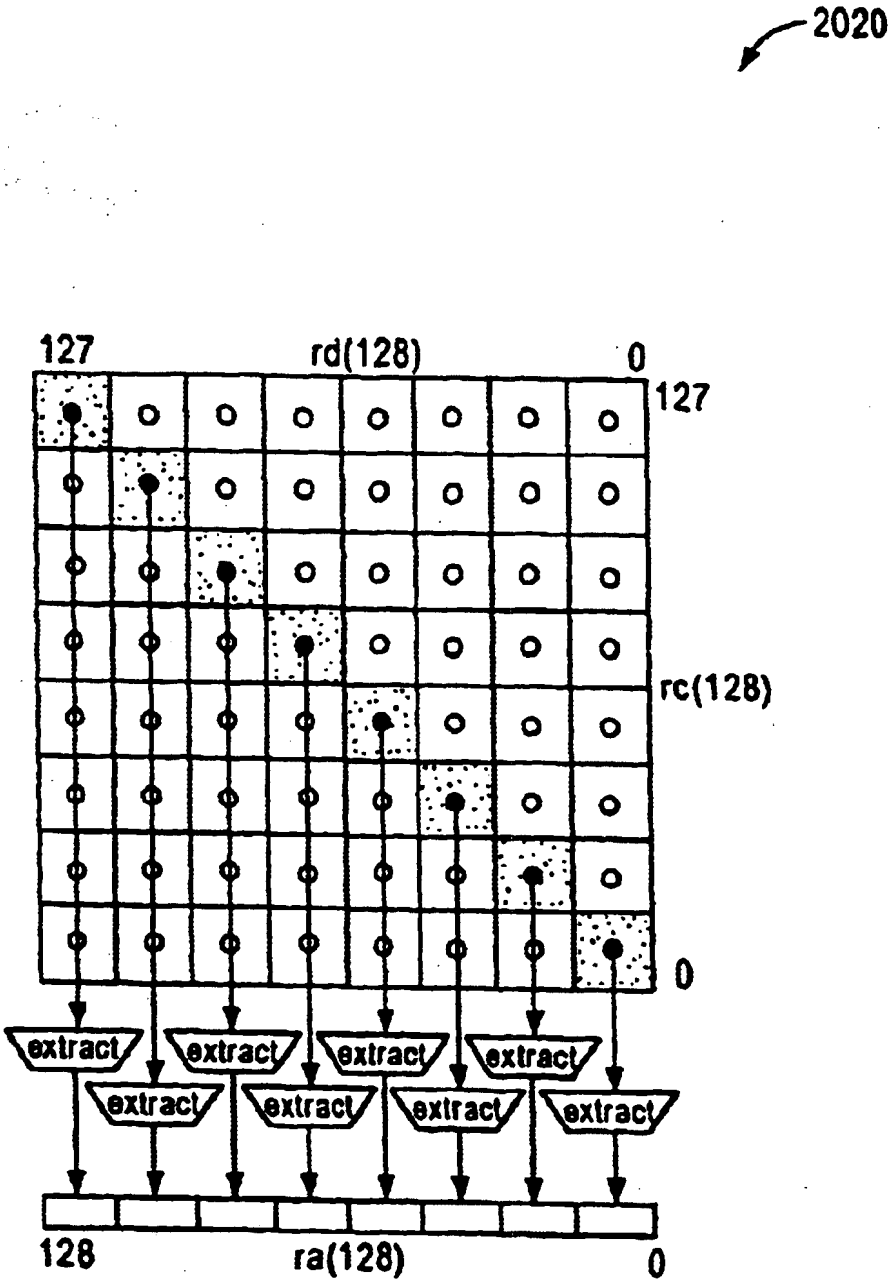
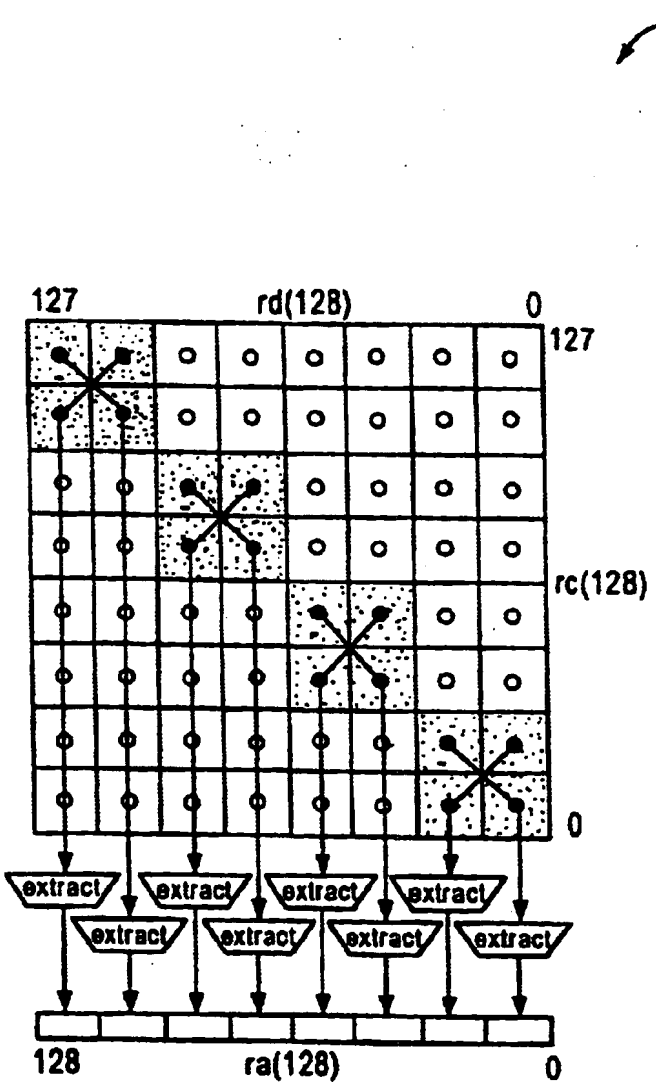

2020
127
rd(128)
0
127
rc(128)
0
extract
extract
extract
extract
extract
extract
extract
extract
extract
128
ra(128)
0

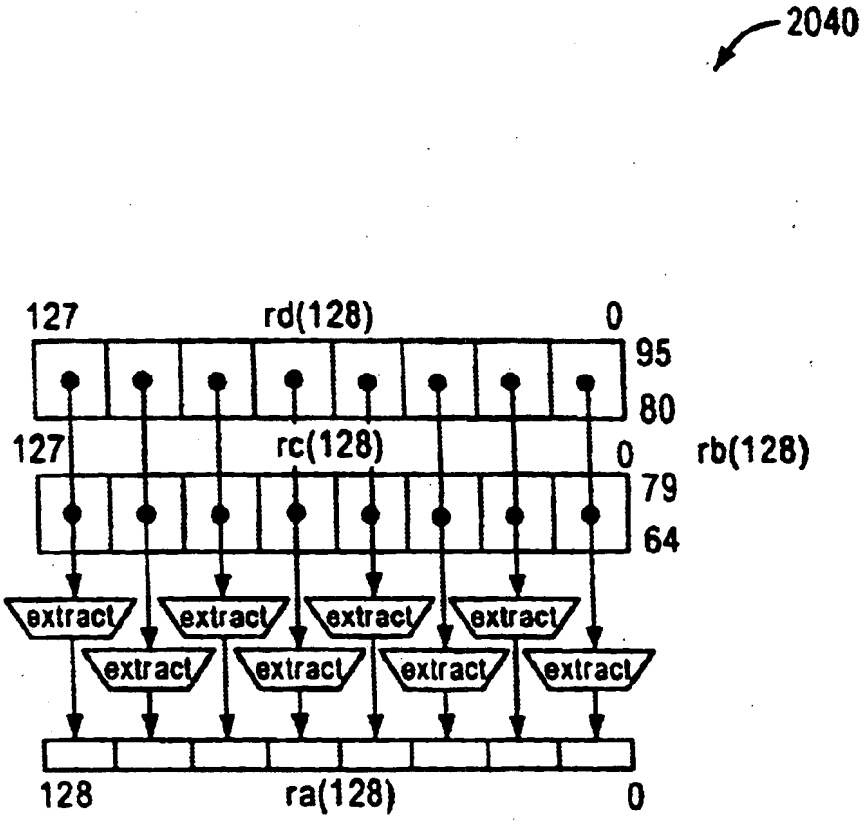
FIG. 20C



Ensemble complex multiply extract doublets

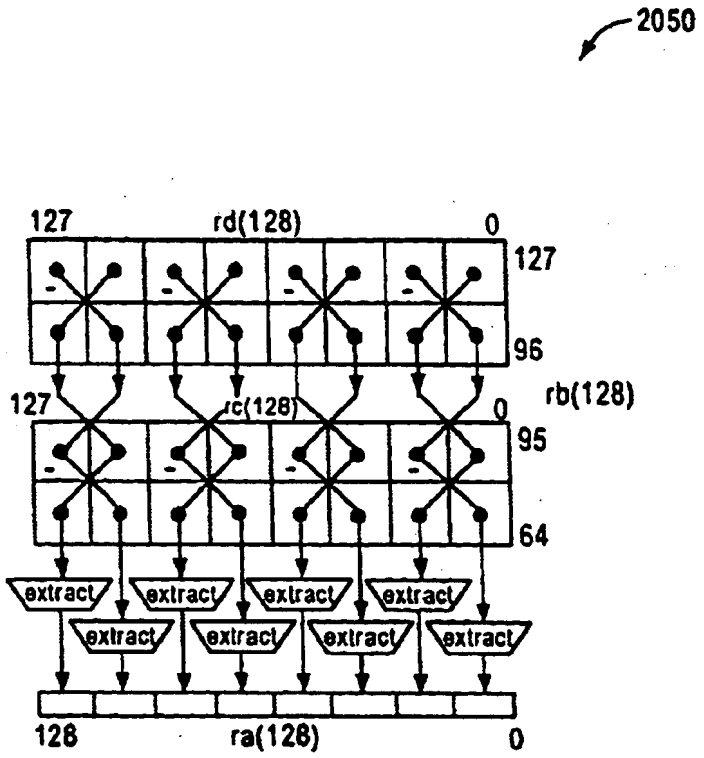
This ensemble-multiply-extract instructions (E.MUL.X), when the x bit is set, multiply the low-order 64 bits of each of the rc and rb registers and produce extended (double-size) results.

FIG. 20D



Ensemble scale add extract doublets

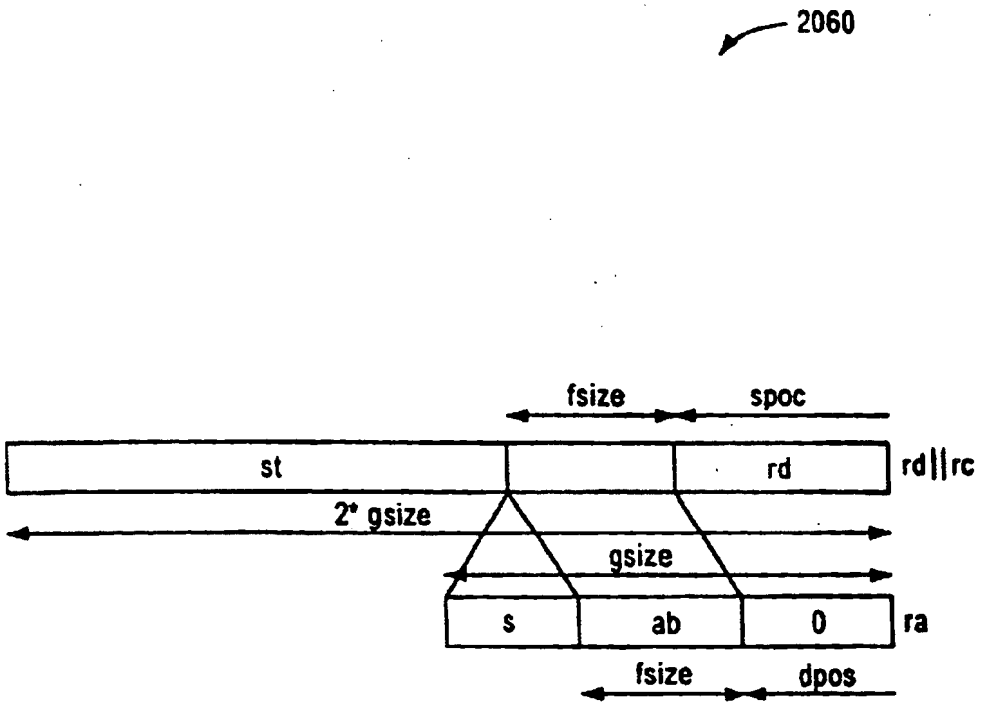
FIG. 20E



Ensemble complex scale add extract doublets

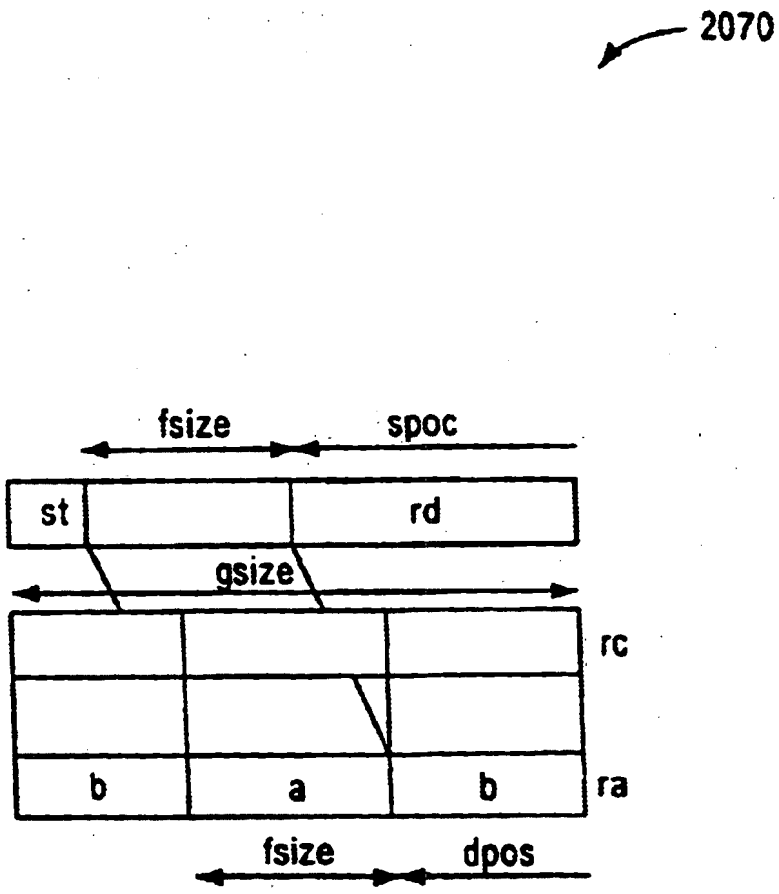
The ensemble-scale-add-extract instructions (E.SCLADD.X), when the x bit is set, multiply the low-order 64 bits of each of the rd and re registers by the rb register fields and produce extended (double-size) results.

FIG. 20F



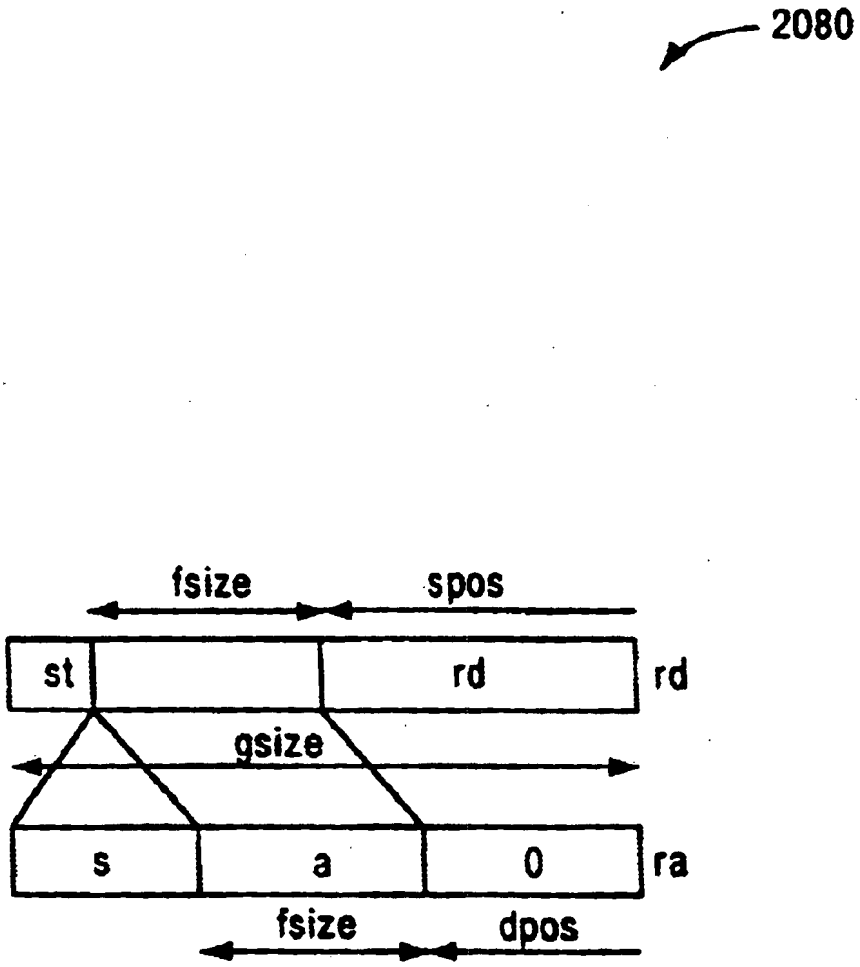
Ensemble extract

FIG. 20G



Ensemble merge extract

FIG. 20H



Ensemble expand extract

FIG. 20I

```
Definition                                                          2090
def mul(size,h,vs,v,i,ws,w,j) as
    mul ← ((vs&v_size-1+i)^h-size || v_size-1+i..i) * ((ws&w_size-1+j)^h-size || w_size-1+j..j)
enddef

def EnsembleExtract(op,ra,rb,rc,rd) as
    d ← RegRead(rd, 128)
    c ← RegRead(rc, 128)
    b ← RegRead(rb, 128)
    case b8..0 of
        0..255:
            sgsize ← 128
        256..383:
            sgsize ← 64
        384..447:
            sgsize ← 32
        448..479:
            sgsize ← 16
        480..495:
            sgsize ← 8
        496..503:
            sgsize ← 4
        504..507:
            sgsize ← 2
        508..511:
            sgsize ← 1
    endcase
    l ← b11
    m ← b12
    n ← b13
    signed ← b14
    x ← b15
    case op of
        E.EXTRACT:
            gsize ← sgsize*2(2-(m or x))
            zsize ← sgsize
            h ← gsize
            as ← signed
            spos ← (b8..0) and (gsize-1)
```

FIG. 20J-1

2090

```
E.SCAL.ADD.X:
    if (sgsize < 8) then
        gsize ← 8
    elseif (sgsize*(n+1) > 32) then
        gsize ← 32/(n+1)
    else
        gsize ← sgsize
    endif
    ds ← cs ← signed
    bs ← signed ^ m
    as ← signed or m or n
    zsize ← gsize*(x+1)
    h ← (2*gsize) + 1 + n
    spos ← (b8..0) and (2*gsize-1)
E.MUL.X:
    if (sgsize < 8) then
        gsize ← 8
    elseif (sgsize*(n+1)*(x+1) > 128) then
        gsize ← 128/(n+1)/(x+1)
    else
        gsize ← sgsize
    endif
    ds ← signed
    cs ← signed ^ m
    as ← signed or m or n
    zsize ← gsize*(x+1)
    h ← (2*gsize) + n
    spos ← (b8..0) and (2*gsize-1)
endcase
dpos ← (0|| b23..16) and (zsize-1)
r ← spos
sfsize ← (0|| b31..24) and (zsize-1)
tfsize ← (sfsize =0) or ((sfsize+dpos) > zsize) ? zsize-dpos : sfsize
fsize ← (tfsize + spos > h) ? h - spos : tfsize
if (b10..9=Z) and not as then
    rnd ← F
else
    rnd ← b
endif
```

FIG. 20J-2

```
for j ← 0 to 128-zsize by zsize                                    ← 2090
    i ← j*gsize/zsize
    case op of
        E.EXTRACT:
            if m or x then
                p ← dgsize*i-1..i
            else
                p ← (d|| c)gsize*i-1..i
            endif
        E.MUL.X:
            if n then
                if (i and gsize) = 0 then
                    p ← mul(gsize,h,ds,d,i,cs,c,i)-
mul(gsize,h,ds,d,i+gsize,cs,c,i+gsize)
                else
                    p ←
mul(gsize,h,ds,d,i,cs,c,i+gsize)+mul(gsize,h,ds,d,i,cs,c,i+gsize)
                endif
            else
                p ← mul(gsize,h,ds,d,i,cs,c,i)
            endif
        E.SCAL.ADD.X:
            if n then
                if (i and gsize) = 0 then
                    p ← mul(gsize,h,ds,d,i,bs,b,64+2*gsize)
                        + mul(gsize,h,cs,c,i,bs,b,64)
                        - mul(gsize,h,ds,d,i+gsize,bs,b,64+3*gsize)
                        - mul(gsize,h,cs,c,i+gsize,bs,b,64+gsize)
                else
                    p ← mul(gsize,h,ds,d,i,bs,b,64+3*gsize)
                        + mul(gsize,h,cs,c,i,bs,b,64+gsize)
                        + mul(gsize,h,ds,d,i+gsize,bs,b,64+2*gsize)
                        + mul(gsize,h,cs,c,i+gsize,bs,b,64)
                endif
            else
                p ← mul(gsize,h,ds,d,i,bs,b,64+gsize) + mul(gsize
                    ,h,cs,c,i,bs,b,64)
            endif
    endcase
```

FIG. 20J-3

```
        case rnd of                                                    ← 2090
            N:
                s ← 0^(h-r) || ~p_r || p_r^(r-1)
            Z:
                s ← 0^(h-r) || p_(h-1)^r
            F:
                s ← 0^h
            C:
                s ← 0^(h-r) || 1^r
        endcase
        v ← ((as & p_(h-1)) || p) + (0 || s)
        if (v_(h..r+fsize) = (as & v_(r+fsize-1))^(h+1-r-fsize)) or not (l and (op =
                    E.EXTRACT)) then
            w ← (as & v_(r+fsize=1))^(zsize-fsize-dpos) || v_(fsize-1+r..r) || 0^dpos
        else
            w ← (s ? (v_h || ~v_h^(zsize-dpos-1)) : 1^(zsize-dpos)) || 0^dpos
        endif
        if m and (op = E.EXTRACT) then
            z_(zsize-1+j..j) ← c_(asize-1+j..dpos+fsize+j) || w_(dpos+fsize-1..dpos) ||
                               c_(dpos-1+j..j)
        else
            z_(zsize-1+j..j) ← w
        endif
    endfor
    RegWrite(ra, 128, z)
enddef
```

FIG. 20J-4

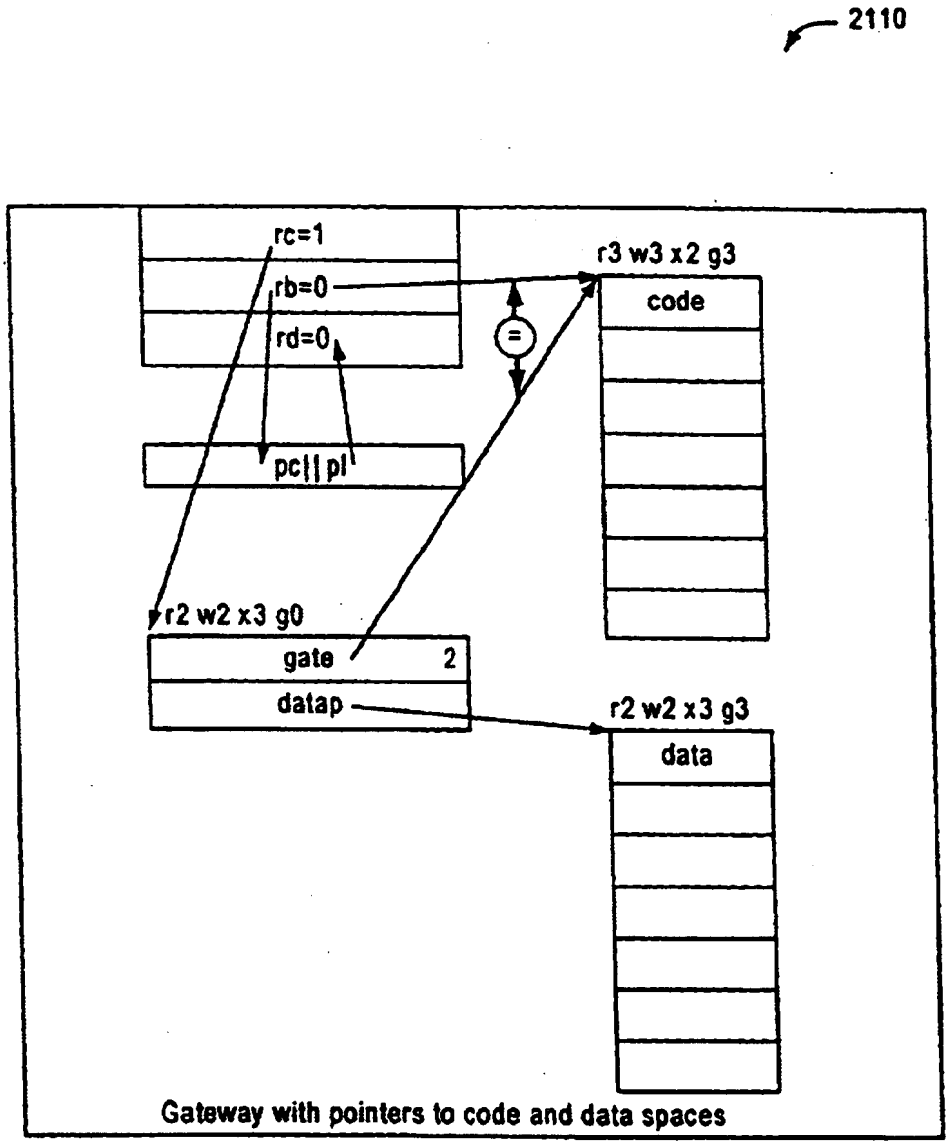

2110
rc=1
rb=0
rd=0
pc|| pl
r3 w3 x2 g3
code
=
r2 w2 x3 g0
gate 2
datap
r2 w2 x3 g3
data
Gateway with pointers to code and data spaces

FIG. 21A

2130

Typical dynamic-linked, inter-gateway calling sequence:
caller:

```
caller   AA.DDI          sp@-size      // allocate caller stack frame
         S.I.64.A        lp,sp,off
         S.I.64.A        dp,sp,off
         ...
         L.I.64.A        lp=dp,off     // load lp
         L.I.64.A        dp=dp,off     // load dp
         B.GATE
         L.I.64.A        dp,sp,off
         ...(code using dp)
         L.I.64.A        lp=sp,off     // restore original lp register
         A.ADDI          sp=size       // deallocate caller stack frame
         B               lp            // return
```

callee (non-leaf):

```
calee:   L.I.64.A        dp=dp,off     // load dp with data pointer
         S.I.64.A        sp,dp,off
         L.I.64.A        sp=dp,off     // new stack pointer
         S.I.64.A        lp,sp,off
         S.I.64.A        dp,sp,off
         ...(using dp)
         L.I.64.A        dp,sp,off
         ...(code using dp)
         L.I.64.A        lp=sp,off     // restore original lp register
         L.I.64.A        sp=sp,off     // restore original sp register
         B.DOWN          lp
```

callee (leak, no stack):

```
callee:  ...(using dp)
         B.DOWN          lp
```

FIG. 21B

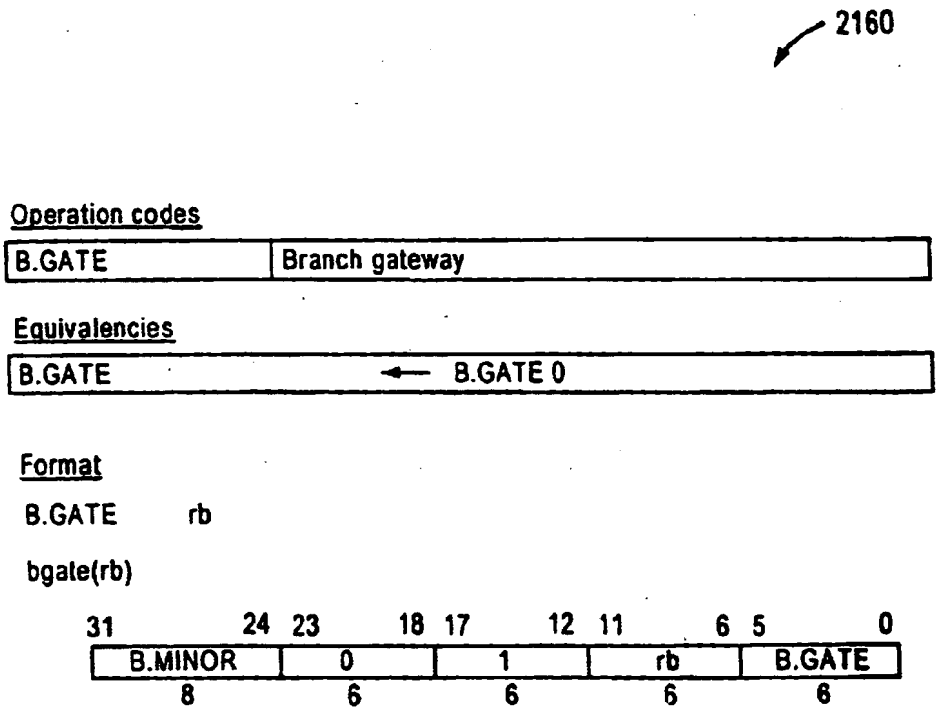

2160
Operation codes
B.GATE
Branch gateway
Equivalencies
B.GATE
B.GATE 0
Format
B.GATE rb
bgate(rb)
31 24 23 18 17 12 11 6 5 0
B.MINOR
0
1
rb
B.GATE
8 6 6 6 6

FIG. 21C

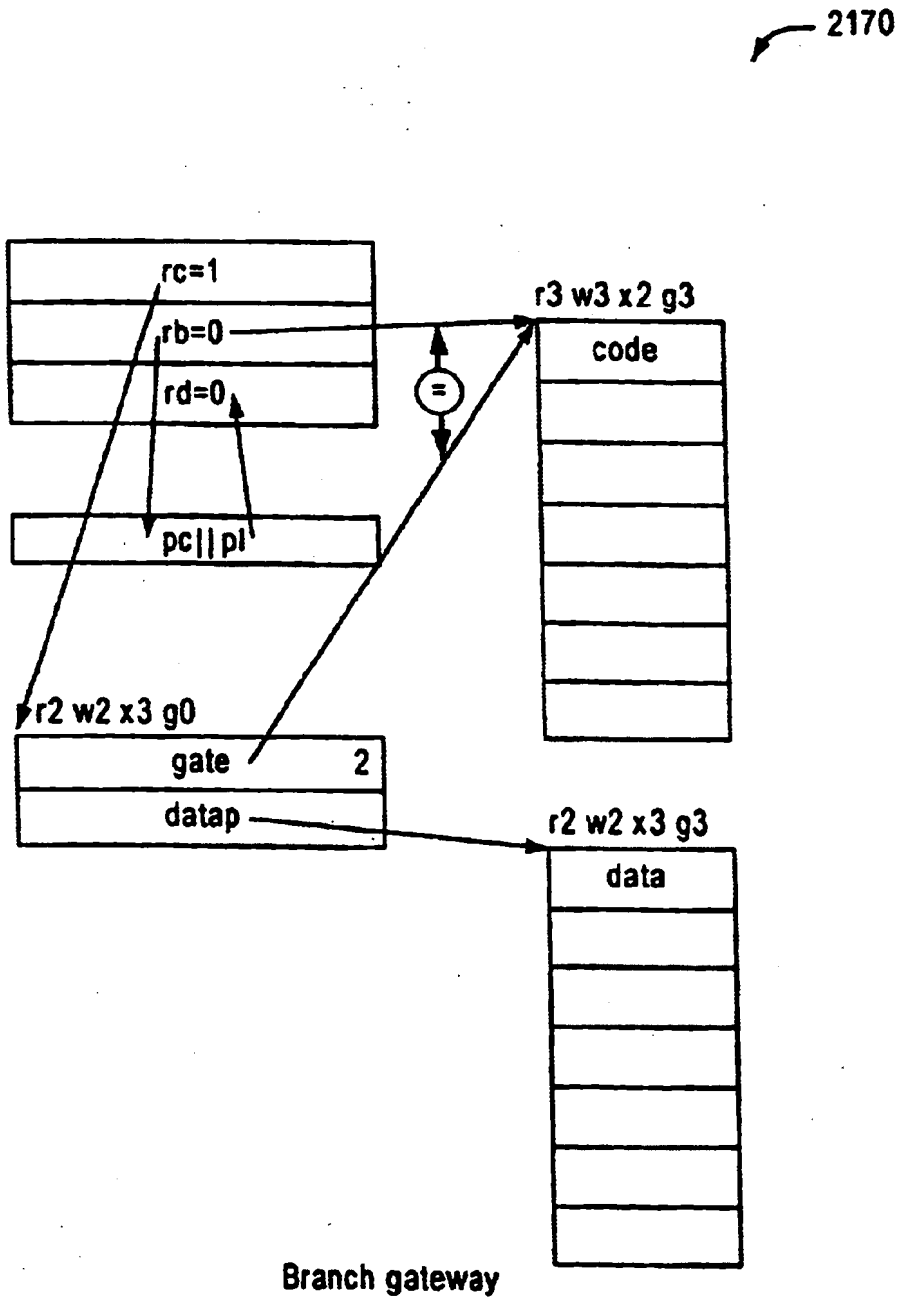

2170
rc=1
rb=0
rd=0
pc||pl
=
r3 w3 x2 g3
code
r2 w2 x3 g0
gate 2
datap
r2 w2 x3 g3
data
Branch gateway

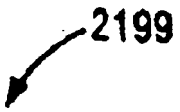
FIG. 21D

2190

Definition

```
def BranchGateway(rd,rc,rb) as
    c ← RegRead(rc, 64)
    b ← RegRead(rb, 64)
    if (rd ≠ 0) or (rc ≠ 1) then
        raise ReservedInstruction
    endif
    if c2..0 ≠ 0 then
        raise AccessDisallowedByVirtualAddress
    endif
    d ← ProgramCounter63..2+1 || PrivilegeLevel
    if PrivilegeLevel < b1..0 then
        m ← LoadMemoryG(c,c,64,L)
        if b ≠ m then
            raise GatewayDisallowed
        endif
        PrivilegeLevel ← b1..0
    endif
    ProgramCounter ← b63..2 || 02
    RegWrite(rd, 64, d)
    raise TakenBranch
enddef
```

FIG. 21E



Exceptions

Reserved Instruction
Gateway disallowed
Access disallowed by virtual address
Access disallowed by tag
Access disallowed by global TB
Access disallowed by local TB
Access detail required by tag
Access detail required by local TB
Access detail required by global TB
Local TB miss
Global TB miss

FIG. 21F

2210

Operation codes

E.SCAL_ADD.F.16	Ensemble scale add floating-point half
E.SCAL_ADD.F.32	Ensemble scale add floating-point single
E.SCAL_ADD.F.64	Ensemble scale add floating-point double

Selection

class	op	prec
scale add	E.SCAL.ADD.F	16 32 64

Format

E.op.prec ra=rd,rc,rb

ra=eopprec(rd,rc,rb)

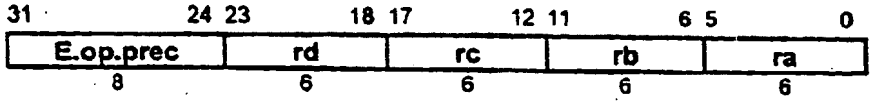


FIG. 22A

2230

Definition

```
def EnsembleFloatingPointTernary(op,prec,rd,rc,rb,ra) as
    d ← RegRead(rd, 128)
    c ← RegRead(rc, 128)
    b ← RegRead(rb, 128)
    for i ← 0 to 128-prec by prec
        di ← F(prec,di+prec-1..i)
        ci ← F(prec,ci+prec-1..i)
        ai ← fadd(fmul(di, F(prec,bprec-1..0)), fmul(ci, F(prec,b2*prec-1..prec)))
        ai+prec-1..i ← PackF(prec, ai, none)
    endfor
    RegWrite(ra, 128, a)
enddef
```

FIG. 22B

2310

Operation codes

G.BOOLEAN	Group boolean

Selection

operation	function (binary)	function (decimal)
d	11110000	240
c	11001100	204
b	10101010	176
d&c&b	10000000	128
(d&c)|b	11101010	234
d|c|b	11111110	254
d?c:b	11001010	202
d^c^b	10010110	150
~d^c^b	01101001	105
0	00000000	0

Format

G.BOOLEAN rd@trc,trb,f

rd=gbooleani(rd,rc,rb,f)

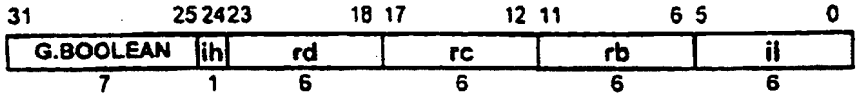


FIG. 23A

2320

```
if f6=f5 then
        if f2=f1 then
                if f2 then
                        rc ← max(trc,trb)
                        rb ← min(trc,trb)
                else
                        rc ← min(trc,trb)
                        rb ← max(trc,trb)
                endif
                ih ← 0
                il ← 0 || f6 || f7 || f4 || f3 || f0
        else
                if f2 then
                        rc ← trb
                        rb ← trc
                else
                        rc ← trc
                        rb ← trb
                endif
                ih ← 0
                il ← 1 || f6 || f7 || f4 || f3 || f0
        endif
else
        ih ← 1
        if f6 then
                rc ← trb
                rb ← trc
                il ← f1 || f2 || f7 || f4 || f3 || f0
        else
                rc ← trc
                rb ← trb
                il ← f2 || f1 || f7 || f4 || f3 || f0
        endif
endif
```

FIG. 23B

2330

Definition

```
def GroupBoolean (ih,rd,rc,rb,il)
    d ← RegRead(rd, 128)
    c ← RegRead(rc, 128)
    b ← RegRead(rb, 128)
    if ih=0 then
        if il5=0 then
            f ← il3 || il4 || il4 || il2 || il1 || (rc>rb)2 || il0
        else
            f ← il3 || il4 || il4 || il2 || il1 || 0 || 1 || il0
        endif
    else
        f ← il3 || 0 || 1 || il2 || il1 || il5 || il4 || il0
    endif
    for i ← 0 to 127 by size
        ai ← f(di||ci||bi)
    endfor
    RegWrite(rd, 128, a)
enddef
```

FIG. 23C

2410

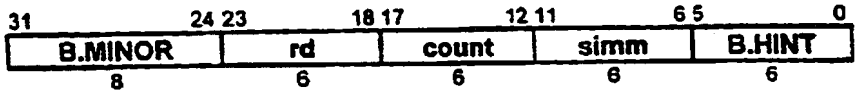
Operation codes

B.HINT	Branch Hint

Format

B.HINT badd,count,rd

bhint(badd,count,rd)



simm ← badd-pc-4

FIG. 24A